Budapest University of Technology and Economics
Doctoral School of Informatics

Novel Network Optimization and Data Compression Methods

Máté Imre Nagy
M.Sc. in Electrical Engineering

PhD Dissertation

Advisor:

Gábor Rétvári, Ph.D. (Senior Research Fellow)

*Dept. of Telecommunications and Media Informatics and
Hungarian Academy of Science Future Internet Research Group,
Budapest University of Technology and Economics*

Budapest, Hungary
2022

“Except our own thoughts, there is nothing absolutely in our power.”

— *René Descartes*

Abstract

It is the nature of industrial evolution that visions and demands of business and marketing sometimes precede what current technology could offer. The unlimited-availability and high-speed access to network services are becoming a real and indisputable need for companies offering real-time applications such as on-demand video streaming or remotely controlled robotic systems. However, the Internet Protocol (IP) is historically not prepared for providing ultra-fast recovery in case of network failures. Although commonly available routing protocols offer built-in healing capability, the services demanding zero downtime still need to look for alternative solutions that usually impose management overhead on the operators.

Focusing on this problem IETF founded the IP Fast ReRoute Framework that is based on two principles: local re-routing and pre-computed secondary next-hops. Unfortunately, the only protection scheme that most router vendors implement is the Loop-Free Alternates (LFA) mechanism that, however, does not guarantee full protection on its own. The first objective of the Dissertation is to present novel network optimization techniques that help to maximize the level of LFA protection without interfering with the default routes. The first approach augments the network with complement edges under the model of jointly failing edges, while in the second round, we make use of the router virtualization technique to be able to deploy a protective overlay on top of the physical network.

In addition to reliable transfer of data, the need for fast processing is also on the rise as the capacity of modern chipsets does not keep up with the upward trend of data growth. Therefore, researchers turned their attention to succinct representations that encode the data with low memory footprint and allow fast operations right on the compressed form. Any compression scheme, however, requires to build and maintain an auxiliary index into the useful data that often outweigh the encoded input itself.

The second objective of the Dissertation is to present a novel, succinct data structure that achieves compression on the data and the index at the same time. Our proposal takes advantage of the block size independent encoding of the input and outperforms the best-known implementations at the price of a slight increase in the storage size.

Kivonat

Az ipari fejlődés természetes velejárója, hogy az üzleti és eladási szektor követelményei megelőzik az aktuális technológiák által nyújtott lehetőségeket. A korlátlan rendelkezésre állás, valamint a gyors hozzáférhetőség a hálózati szolgáltatásokhoz alapvető igénnyé vált az olyan alkalmazásfejlesztők részéről, akik például élő videófolyam-továbbítással vagy távoli robotrendszerek iranyításával foglalkoznak. Historikus okoknak köszönhetően azonban az IP protokoll nincs felkészítve a hálózati hibák rendkívül gyors helyreállítására. Noha az elérhető útválasztó protokollok beépített javítási képességekkel rendelkeznek, azon szolgáltatások melyek képtelenek bármiféle infrastruktúra hibát elviselni továbbra is alternatív megoldások felé kell tekintsenek, melyek általában jelentős kezelhetőségi többletköltséggel járnak.

Ezen problémát szem előtt tartva az IETF megalkotta az IP Fast ReRoute keretrendszert, melynek két fő alapelve a biztonsági csomópontok előre történő meghatározása, valamint a forgalom lokális átirányítása. Sajnos az egyetlen védelmi mechanizmus mely a legtöbb útválasztó termékben elérhető, a hurokmentes elkerülőutak módszere (LFA), önmagában nem nyújt teljes védettséget. Az Disszertáció első célkitűzése, hogy olyan újszerű hálózat-optimalizálási módszereket mutasson be melyek anélkül képesek maximalizálni az LFA védelem mértékét, hogy megzavarnák az eredeti csomagtovábbítási útvonalakat. Az első megközelítésben a hálózatot kiegészítő élekkel látjuk el, figyelembe véve a közösen meghibásodó élek lehetőségét, majd ezt követően az útválasztókon elérhető virtuális példányokat használjuk fel, hogy védőhálót terítsünk a fizikai topológia fölé.

A megbízható adattovábbításon kívül az azok gyors feldolgozására történő igény is erősödik, hiszen a modern csipek kapacitása nem képes lépést tartani a keletkező adatok mennyiségének növekedésével. A kutatók ezért figyelmüket az adatok egy olyasféle

tömör ábrázolására irányították, mely amellett hogy kisebb memóriaigénnyel rendelkezik, egyben támogatja az olyan gyors műveletek elvégzését is melyek a tömörített adaton közvetlenül dolgoznak. Bármely tömörítési technika ugyanakkor egy külső index meglétét feltételezi a hasznos adat címzésére, amely azonban gyakran képes méretben meghaladni magát a hasznos adatot. A Disszertáció második célkitűzése egy újszerű tömörített adatstruktúra bemutatása, mely nem csak a hasznos adat de egyben az index méretén is képes csökkenteni. A javaslatunk a blokkméret független kódolás előnyeit használja fel, és kicsivel nagyobb tárolási méret árán felülmúlja teljesítményben a ma ismert legjobb tömör ábrázolási formákat.

Acknowledgements

For some reason I was lucky enough to grow up in a family who never stopped encouraging me to aim high and to give my best to reach my goals. It is not just the way I was treated makes me thankful, but also that I could witness my Parents and Zsófi, my sister, to set a great example of how to live a happy and meaningful life, for which I am extremely grateful. Along the way of education, my teachers earned the deepest respect and gratitude for being able to patiently share their knowledge, starting from the Piarist Grammer School all the way to BME, and more specifically to the High-Speed Networks Laboratory.

My warmest thanks are due to Gábor Rétvári, my supervisor, who never needed more than five minutes to fire me with enthusiasm about research topics he worked with. His passionate curiosity for discovering, new, unexplored problems and the selfless attitude to guide strangers in the unknown revealed the hiding beauty behind dull numbers or algorithms. Needless to say that without his continuous support and generosity this work would have never been able to come to its end. I must also thank to János Tapolcai for his advice in my math and publication related questions, and for the invitation to the Lendület community. The list of my mentors would not be complete without Péter Bakki, my uncle, who has spared no effort to help me to become an engineer and who never refused to explain a tricky algorithm, or to fix a desperately crashing program code.

I also owe thanks to all of my colleagues at Ericsson IMS Gateways, who helped and accompanied me in the exploration of the deep rabbit hole of telecommunication systems. Special thanks goes to Mika Raitala, Zoltán Boncz, Dávid Tisza and Dietmar Fiedler for their trust and open-minded, knowledge-sharing attitude, and specifically for teaching me how to properly produce and eliminate handcrafted beers. I wish to say thank you to Ericsson Research for the inspiring atmosphere, and in particular

to Péter Mátray for his constant encouragement. Furthermore, I must express my deepest gratitude to my managers, Tamás Dávid, Zoltán Csapó and András Császár, who gave me the opportunity to bring out my best during this long journey.

Last but not least, I wish to thank all my friends for not letting me silently give up this work by their relentless inquiry on the expected date of finish. I am very grateful to Bea, Isti, András, Gábor, Ádám, Gergő, Ubi and Péter Nagy for making me laugh in unexpected or desperate moments, and to Ales, Luiza, Mariscal, Pepe, Attila and Matthias for showing me new cultures and for keeping my soul young and free.

Contents

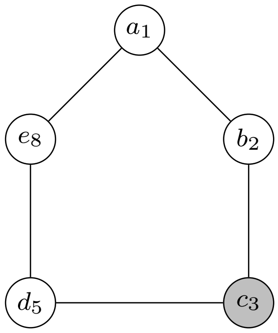
Chapter 1

Introduction

It comes as no surprise that the Internet is growing [1]. At the moment, inconceivable amount of data is traveling on the wires or being transmitted by antennas. Those bits that are not on their way are waiting in CPU registers or memory for being processed while yet others are actually under processing. This whole complex and well designed system, called the Internet, has one simple goal: making human being's life easier. The purpose is so well served that society shifts more and more services to the digital ecosystem where expectations on *availability* and *speed* are constantly on the rise.

Today, thanks to the appearance of smartphones, we all have daily hands-on experience about such services like telephony, instant messaging or video streaming [2]. However, any system built to enable communication between endpoints has major challenges to face. A well-known example is video streaming which often annoys the user with sudden decrease in quality (e.g. due to limited bandwidth) or frequent disconnections caused by malfunctioning network devices. Therefore, throughout the entire lifetime of a network, the main goals of the operators are **(I) to mitigate the possible failure events** in the infrastructure and **(II) to ensure that all devices operate with maximum efficiency**.

First of all, getting prepared for arbitrary network failures is already a complex task on its own. The reason lies behind the so-called "all-IP" phenomenon [3], where all competing transfer protocols are flattened into a single packet switching mechanism. However, the Internet Protocol (IP), is built to work in an unreliable, called "best-effort" manner that poses difficult challenges to operators when they try to guarantee the five nines (99.999%) availability. The distributed system, managed



(a) Sample topology

dest.	nh.	LFA
–	d	–
a	b	d
b	b	–
d	d	–
e	d	b

(b) Forwarding table of c

Figure 1.1: Demonstration of Loop-Free Alternates in a selected topology.

by routers, can easily get into an inconsistent state where forwarding tables of routing devices show contradicting entries due to failures in the network.

The concern about inconsistency roots in the fact that these entries sometimes create a forwarding loop that makes packets bouncing between the interfaces of devices that eventually eats up all their processing capacity. The remedy for this phenomenon is to stop handling all traffic until the routers resolve the situation by a so-called restoration process [4, 5]. As soon as it is finished, the traffic is good to go — but only if it survived the downtime at all. This can often take hundreds of milliseconds [6] that is usually unacceptable for highly-sensitive traffic.

To overcome this problem, the networking community started to seek for alternative solutions that reduce the recovery time of IP networks below tens of milliseconds [7]. The idea is to get prepared for the failures before they could occur by *proactively calculating secondary paths to each destination, and to re-route the traffic to the alternate path until the restoration is finished in the background.* The challenge here is to find alternates that guarantee the loop-free delivery of packets during this inconsistent state of the network. One particular solution, called the *Loop-Free Alternate*, Loop-Free Alternate (LFA) [8], emerged lately from the wide variety of proposals.

Let us overview this concept through the example depicted in Fig. 1.1a. The network consists of five routers and links of uniform cost. Each device is labeled with a letter and a number (will be used later). We assume that packets follow the shortest possible path between two nodes (e.g. from c to a it is the c-b-a path). For a brief moment, let us imagine that the link between c and b becomes unavailable.

As we claimed before, the $c \to a$ traffic gets stopped until the restoration of the network is finished, or unless we find a proper secondary path to which c can redirect its packets. Luckily, it is easy to notice that regardless the failure of link c-b the neighboring router, d, is still able to deliver packets to a along the path d-e-a. And most importantly, this path is guaranteed to be loop-free as it does not contain the source of the packet, c, so there is no risk of triggering a ping-pong effect between c and d.

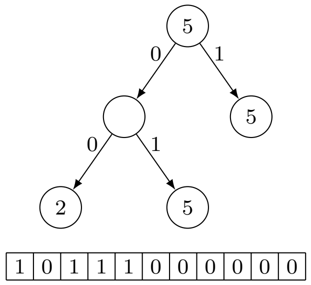
The list of LFAs for each destination is available on Fig. 1.1b. One can easily spot the incomplete entries in the last column which translates as missing alternates for certain destinations. The ratio of protected versus all destinations is only 50% that is not particularly satisfying, especially for operators running live services. Therefore, the first important goal of the Dissertation is to raise the level of LFA protection by recognizing and taking advantage of the dependency between the number of protected node pairs and the layout of the topology. Accordingly, *we formulate different network optimization problems and we design novel algorithms incorporating the most common failure models in order to bring LFA protection perfect in communication networks.*

Our intention to increase the efficiency of networking related data processing is not in the least easier task either. Not just the growth in scale of data, but also the patterns and types of queries went through significant changes lately. Obviously, the way of representing information followed these changes that gave rise of a wide set of novel data structures, but without never losing sight of the ultimate goal: *providing the fastest possible answer by consuming the smallest amount of storage space.* This is because the speed of access is mainly defined by the location of the data in the cache hierarchy. Therefore, keeping the size of these structures small is essential to avoid — the otherwise necessary — costly CPU disk and memory accesses that can manifest in degraded end-user experience.

A practical example is the poorly performing routing device that can hugely add to the overall latency of media streams or introduce jitter in the network. To demonstrate this problem, we switch to a more realistic representation of network addresses that is based on ids, and so we present the corresponding forwarding table of router c in (Fig. 1.2a). The prefix column shows how to merge multiple rules by using the binary format of destination addresses and the wildcard character ($*$) to match any

dest.	prefix	nh.	LFA
-	*	5	2
1,2	00*	2	5
5	0101	5	2
8	1*	5	2

(a) Forwarding table



(b) Prefix tree

Figure 1.2: Demonstration of forwarding table compression.

bits. Thanks to this trick, the number of rules is now reduced to 4, but we still need to step through all the rules if the entry for the destination of a packet resides in the last row of the table.

Since this is rather inefficient, the networking community turned to the prefix-tree representation [9] of the forwarding table that is available on Fig. 1.2b. The prefix-tree is a binary-tree that is "forwarding equivalent" with the table, meaning that by following the branches of the tree leads us to the exact same output — except that in fewer steps than the table. Undoubtedly, the tree gives a more simple and eye-catching overview on the rules, but also raises the question of how to encode it binary? Among the several possible encoding schemes, we show the level-order encoding that writes 1 for a node holding value and 0 otherwise [10].

Accordingly, we write 1 for the root node as it holds the id of the default next-hop, 5. The next layer consists an empty node and a node with value, therefore we append $'01'$ to the bitvector. Also, the algorithm assumes that nodes holding values are never leaves, so every time such a node is reached we insert $'00'$ as if it would have two empty children. This explains the encoding of the third ($'1100'$) and the fourth layers ($'0000'$). The resultant encoded bitvector is shown on Fig. 1.2b.

The encoding reflects the structure of the tree, and we store the symbols alongside in an array. This requires only 11 bits, but *to be able to answer queries on the encoded format some special operations are needed.* Since the layout of the structure is defined by the position of 0s and 1s in the bitvector, we must support queries like "what is the position of the i-th 0?" or "return the number of 1s until the i-th position". Researchers found that the bitvector can be freely compressed as long as they are still able to provide fast answers for these types of queries. However, space reduction and fast access to the data require several trade-offs. Consequently, the last goal of the Dissertation is to propose *a novel succinct data structure that offers noticeable performance gain in queries compared to the state-of-the-art implementations while achieves entropy-constrained space reduction on the storage size.*

Chapter 2

LFA Graph Extension under Correlated Failures

This chapter is devoted to introduce IP as a network level protocol and describe the main principles of route selection. We introduce the IP Fast ReRoute Framework (IPFRR) and the most popular IPFRR proposal, the Loop-Free Alternates. Then we present the LFA Graph Extension problem under correlated failures that asks for adding the minimum set of complement edges to raise the LFA coverage to 100%. Finally, we carry out several measurements and evaluate our work.

2.1 Background

In general, we could say that all types of communication is based on *protocols*. The protocol is a well-defined set of rules that is commonly used and accepted by each communicating party. In the case of the Internet we use several protocols, each with a different purpose. This family of protocols is often referred as the Internet Protocol suite, also known as the TCP/IP (Transmission Control Protocol/Internet Protocol) model [11]. As it can be seen on Fig. 2.1 they are grouped into four levels. Perhaps the easiest way to understand the role of these levels is through an example. Let us suppose on Fig. 2.1 that the host would like to access the content of a remote web page. For this purpose, it constructs an *Application* layer level message that in our case is a HTTP (HyperText Transfer Protocol) request containing the details of the desired webpage. This is then passed to the lower - *Transport* layer. As the

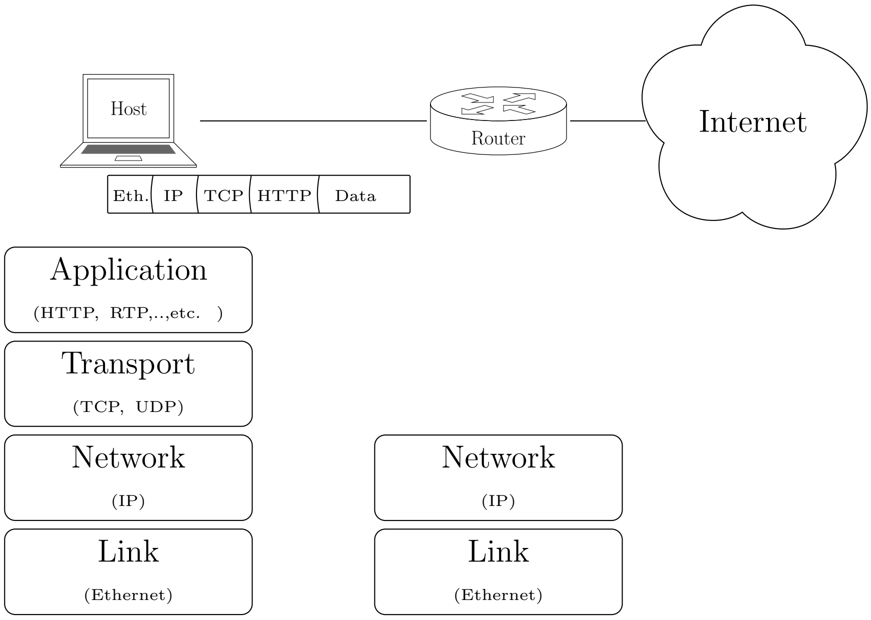


Figure 2.1: The TCP/IP protocol stack and an example for message encapsulation. Note that routers only use the Network level IP addresses for making forwarding decision. The above layers are invisible for routing.

naming implies, this one is responsible for transport reliability. If the application needs guarantee that all the sent bytes indeed arrive at the receiving side, then TCP is used while if loss of data is acceptable during delivery, it selects the User Datagram Protocol (UDP).

Since HTTP is based on TCP, after appending the corresponding header to the message, it is given to the underlying *Network* layer. Here the packet gets extended by two IP addresses. The IP address is a network level identifier that always varies depending on the location from where the device is connected to the Internet. Furthermore, the IP addressing scheme makes possible to easily aggregate and exchange *prefixes* that is the group of possible addresses a domain can have. Returning to our example, the Network layer extends the packet with the source ("Host") and destination IP addresses ("Web server") and gives the composed structure to the very bottom *Link* layer. This finally performs the physical transmission of the packet that is then arrives to the adjacent network element[1].

In the literature this process is often called *encapsulation*, while the reverse, when

[1] The Link layer also extends the packet with a globally unique (MAC) address, but we omit the details for brevity.

the receiver decomposes the incoming message is the *decapsulation*, respectively. In between, during their travel through the network, packets are forwarded in a hop-by-hop manner. At each hop *routers* direct the incoming packets to the proper interface based on their internal database that maps prefixes to network interfaces. Note that the forwarding decision is solely based on the Network layer leaving the fields of upper layers intact during the delivery (Fig. 2.1).

Before continuing the discussion of different types of networks and routings, a short stop is necessary here. Recall, that up to now the reliability was only guaranteed by the Transport layer that is completely ignored in forwarding. In fact, TCP operates by tracking and retransmitting lost packets what is the best it can do on top of the unreliable IP protocol that works in a *best effort* way. Unreliable delivery is still perfect for media streams (e.g. video, audio) that can live with a small ratio of packet loss as it only manifests in slightly degraded media rather than complete disruption.

The Internet is organized in a hierarchical structure. The set of network elements that are owned and administered by a single entity (e.g. an enterprise or educational network) is called an Autonomous System (AS). Within an AS, routers exchange forwarding information with each other, but any time a packet is destined to an external address they get passed to a dedicated *egress router*. Fig. 2.2 shows a high-level overview of the Internet routing system.

Both the routing and the way how routes are advertised inside and outside of an AS are completely different. First, within an AS the routing information is exchanged via the intra-domain, or so-called Interior Gateway Protocols (IGPs), such as Open Shortest Path First (OSPF [4]) or IS-IS (Intermediate System to Intermediate System [5]). They perform shortest path routing based on the undirected graph that models the routing devices and the cabling between them.

One level above the inter-domain routing protocols, like the prevalent Border Gateway Protocol (BGP), not only interconnects the ASes but also allow them to implement their own routing policies that are influenced by economical or political aspects. Typically, the domain of an Internet Service Provider (ISP) can be treated as an AS, and since ISPs are commercial enterprises they have business relationships with each other. In this manner, in addition to provide global reachability to their customers, it is also important to keep their operational expenses low [12] and to hide the details of their network from the connected competitors. BGP is also special in

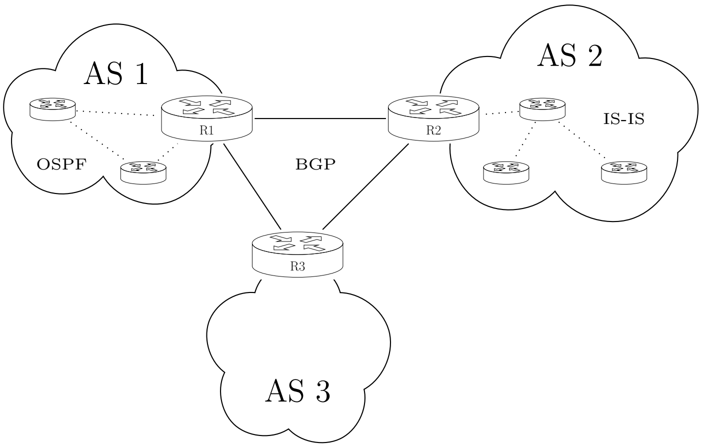


Figure 2.2: The shortest path based OSPF and IS-IS are responsible for the intra-domain routing in AS1 and AS2, while BGP handles the communication within the ASes.

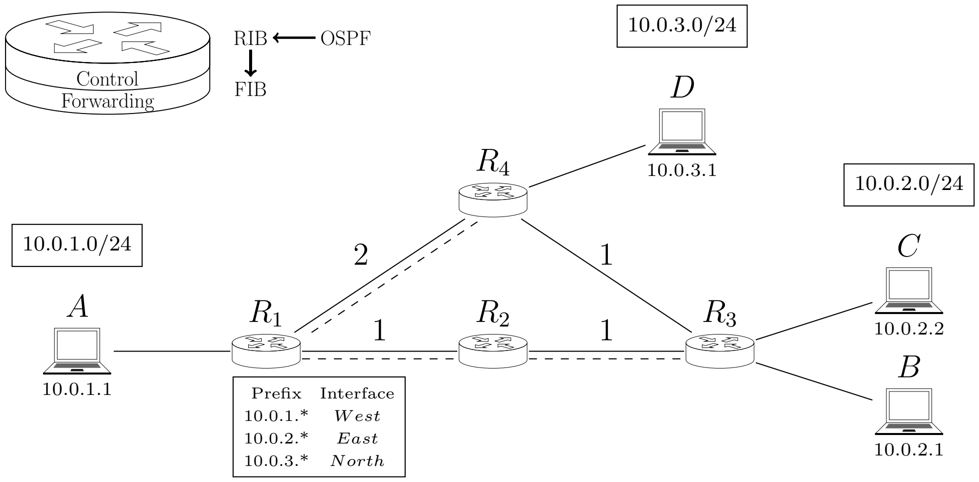
the sense that it is capable to handle enormous amount of addresses for which IGP protocols are not prepared.

Throughout the Dissertation we restrict ourselves to improve the reliability of networks running IGP. Therefore, if it is not stated otherwise, we mean intra-domain routing when we simply refer to routing.

2.1.1 Routing Protocols

Historically, the ancestor of the Internet was built to connect military centers. Correspondingly, the auto-healing capability was among the main goals of the designers when they drew the outline of a *distributed* system. This property then quickly became the main benefit as well as the downside of the system. Think about the main pitfall of distributed systems, consistency, that is inherently hard to enforce within independently working entities. As an example, we can briefly refer to the fluctuating trend of the networking community that alternately favors centralized vs. distributed control (e.g. SDN [13], Software-Defined Networking).

The key component in distributed networking systems is the intelligent router that is able to individually maintain and update the list of available paths in the network. In addition, the Internet Protocol makes extremely easy for applications to communicate with each other as the *destination based routing* principle pushes the logic of packet forwarding entirely to the network. Loosely speaking, the sender can



Prefix	Interface
10.0.1.*	*West*
10.0.2.*	*East*
10.0.3.*	*North*

Figure 2.3: An example for demonstrating IGP routing. In the upper left corner a simplified router architecture is shown with the control layer maintaining the routing information base (RIB) and a data layer that holds a more efficient representation of the forwarding rules (FIB). The network topology connects three LANs. Observe the routing table computed by R_1 and the corresponding shortest path tree (dashed lines).

operate in a “fire-and-forget” fashion if it applies the proper protocol settings on the packet.

A simple forwarding example is shown on Fig. 2.3. As we already mentioned earlier, IGP protocols perform *shortest-path based* forwarding that requires to have *link costs* attached to the connections in order to be able to calculate distances. This single scalar is a rough abstraction for representing different physical parameters such as bandwidth capacity or end-to-end delay. Now, by having costs in place, the goal of the routers is to find and keep track of the available shortest paths to each destination. The algorithm that calculates these paths in a given graph was invented by Dijkstra [14], and since then it is still a popular subject of performance improvement ideas [15].

The network on Fig. 2.3 consists of four routers and three hosts that reside in different Local Area Networks (LAN). For the sake of simplicity, we omit intermediate network segments (i.e. switches). Let us suppose that all routers have finished the calculation of the Dijkstra algorithm and hence they have the exact same view of the topology. If a packet now is sent from host A to B, then it takes the R_1-R_2-R_3 path. Notice in the routing table of R_1 the list of routable network addresses that are mapped to the corresponding interfaces. During the route selection process the router performs the *longest prefix match* against the destination address to find the most specific entry.

In practice, even though a single entry points at network 10.0.2.0, both destination hosts (B and C) are accessible through the same *next-hop* of R_1. Another interesting property of this topology is that by modifying all link costs to uniform, then there would be two equally long paths from A to B. The technique to forward traffic along multiple paths simultaneously is called *Equal Cost Multi-Paths* (ECMPs) that requires the routers to maintain several next-hop entries to the different prefixes. Although it seems appealing from traffic engineering point of view to mitigate the risk of service failures by using parallel paths towards a destination, the downside is the difficult monitoring and the side effect of having packets re-ordered during delivery that introduces jitter [16, 17].

By being aware of the core logic of path selection, it is time to discuss how routers discover and learn the set of forwarding routes. This is exactly what a *routing protocol* does: allows routers to exchange, build and maintain reachability information up to date. Thanks to the routing protocols, routers are able to dynamically detect changes in the network and to react to these changes by recomputing the topology and propagate the updates across the network.

The two types of IGP routing protocols are the i) *distance-vector* routing that is based on the periodic exchange of complete routing tables within the neighbors, while ii) *link-state* protocols flood the network with small routing advertisement fragments that are the main building blocks utilized by routers for calculating the complete view of the topology. Since link-state protocols are dominant in IGP, we continue with discussing their principles.

Everything starts with a short HELLO message that is apart of helping the discovery of adjacent routers it also plays key role in failure detection. As soon as the local discovery gets complete, the router builds up its internal database (LSDB) and shares this information with its neighbors. The process, when the exchange of topology information takes place is often referred as *flooding*, and whenever it is finished the routers become ready to compute the self-rooted shortest path tree (with Dijkstra) that is finally get pushed to the forwarding plane. We call the elapsed time *convergence time* and the topology *converged* at this final state.

A very similar process is executed when a failure occurs in the network with the difference that missing HELLO messages indicate the presence of an incident. This time, however, the delay in distributed route calculation can lead to the creation of forwarding loops. Observe that R_4 reaches R_2 through R_3 in our example (Fig. 2.3). Now if the link between R_2 and R_3 fails, then R_3 updates its routing table well before R_4 (since R_3 is adjacent to the failure) and it marks R_4 as the only possible next-hop towards R_2. Unfortunately, R_4 is not yet notified about the failure so it still considers R_3 as its next-hop to R_2. What happens now is that packets destined to R_2 arriving either to R_4 or R_3 start to bounce between R_4 and R_3 until the expiration of the “Time-to-Live” (TTL) field of the packet. What is worse, in case of continuous traffic, the result is an infinite loop that eats up all the bandwidth and CPU resources of R_3 and R_4 leading to a complete breakdown of the connection. These transient loops, caused by the IGP restoration process, are called *micro-loops* and operators try to avoid the occurrence of this phenomena at all cost in their networks [18].

To eliminate the probability of creating micro-loops, the remedy of the restoration process is to disable any traffic until the network becomes converged. Needless to say, this gap in delivery is unacceptable for certain applications, therefore several proposals emerged in the 2000s to solve this problem. First, we refer to the Multi-Protocol Label Switching (MPLS [19]) that intended to provide a more fine-grained control over route selection of TCP/IP networks. The idea is to assign short labels to the packets that are then inspected and used by the routers for making the forwarding decisions. This technique also aimed to replace the longest-prefix match of IP routing with the more efficient exact match [20] that finally became a marginal problem thanks to the fast evolution of commodity hardware. Nowadays MPLS is mostly used for traffic engineering purposes such as enforcing traffic prioritization for premium customers or to provide protection against certain kind of failures.

Unfortunately, the routing protocol of MPLS (RSVP-TE, Resource Reservation Protocol with Traffic Engineering) can often cause signaling overhead in the network as it is proportional to the number of labeled paths, and also the administration of these labeled paths requires cumbersome steps in the configuration that hinders its overall applicability [21, 22]. Therefore, many operators have not deployed MPLS at all and run their services on top of pure IP networks.

2.1.2 Fast Re-Route Proposals

In the previous section, we gave an insight into some of the problems IP routing struggles with. Our main goal is *to reduce the recovery time of IP networks below tens of milliseconds* by implementing a **proactive** scheme that executes **local** recovery actions.

In order to provide fast recovery for pure IP networks, the Internet Engineering Task Force (IETF) has initiated the IP Fast ReRoute (IPFRR [7]) framework in 2006. The main principle is the computation of backup routes that enables routers to apply local repairing without the need of informing other routers about the failure. This allows to eliminate one of the most time-consuming step of the IGP recovery process, the flooding. Instead, when a network element becomes unavailable, the neighbors immediately switch to the backup routes and traffic flows without major disruption while the IGP converges in the background. Since there is no chance for network elements to exchange any information on the failure, the alternate paths must guarantee that traffic will never be directed into a forwarding loop. There is a colorful variety of approaches that have been presented under the umbrella of IPFRR, however, most of them are struggling to find their way through standardization bodies which makes them less attractive for commercial use. In this section, we briefly go through the list of the most important IPFRR variations.

With *Failure Insensitive Routing* (FIR), routers perform interface specific packet forwarding [23, 24, 25]. If a packet arrives on an unusual input port, the router infers that due to some network element failure the packet has been redirected to a detour at an earlier stage of its forwarding path. Based on this information, it tries to retransmit the packet on an output interface that it knows is not affected by the fault. Unfortunately, in current commercial routers interface specific packet forwarding is not available.

As its name foreshadows, the *Not-via addresses* [26] IPFRR scheme introduces tunneling to route around the failed component. When a router loses contact with one of its neighbours, it encapsulates packets in a new IP header with marking the destination as a special not-via address. The semantics of this not-via address can be translated like “forward me to the destination not-via the failed element”. All the messages affected by the failure are transmitted through this tunnel afterwards.

Unfortunately, computational and management complexity of handling the numerous not-via addresses the proposal requires can be daunting [27]. To reduce these costs, authors in [28], and the subsequent Internet draft [29], introduce a *lightweight version of Not-via*. The idea is based on maximally redundant trees [30, 31], a pair of trees for each node with the useful property that any single link or node failure will leave at least one of the trees intact, under the assumption that at least two disjoint paths existed in the network. Unfortunately, neither Not-via addresses nor Lightweight Not-via have been standardized yet, neither commercially available implementations exist.

Multiple Routing Configurations (MRC, [32]) calculates a small set of backup network configurations (or overlays), maintaining the invariant that for any source-destination pair at least one of the configurations remains connected after any single failure. When a link or node fails, packets are marked with the proper backup configuration identifier that enables routers to use the appropriate overlay topology. Unfortunately, marking packets would consume invaluable bits from the IP header. Another approach, called *O2* routing [33], is to keep two distinct loop-free next-hops towards each destination. In case of link failure affecting one of the next-hops a router can immediately switch packet forwarding to the other one. The disadvantage of this concept is that it breaks shortest path routing, fundamental to IGP operations today.

Protection routing [34] is different from the others in the sense that routers are only responsible for packet forwarding and the routing information is stored in a central server. The server calculates a set of different routing trees, not necessarily coincident with shortest paths, accompanied with a carefully chosen set of secondary next-hops for each node that can be used when the primary next-hop disappears. After downloading precomputed information to the routers, if any next-hop becomes unreachable, then routers can quickly switch to their standby next-hop. The main difficulty is in calculating the routing trees (more precisely, routing DAGs) and the corresponding secondary next-hops to guarantee loop-free forwarding and 100% protection coverage. This problem, similarly to most resilience maximization network optimization problems, turns out NP-complete. Apart from this difficulty, implementing protection routing would also require centralized control, which would mandate a deep reorganization of the way today's IP networks are managed [35].

Last, but not least the *Loop-Free Alternates* builds on the beautifully simple idea that a neighbor is only selected as backup next-hop if its shortest path to the destination does not cross the sender itself. Introducing LFA in an operator's network requires nothing else, but a simple software upgrade in the routers. It does not rely on non-standardized, vendor-specific features or protocols that usually come at the price of increased management overhead. Thanks to this approach LFA has become a standardized and well-tested protection mechanism that is already used widely among most major router vendors [36, 37, 38]. The only problem is that LFA usually does not come with perfect protection as it strongly depends on the layout of the topology. Moreover the level of protection varies between failure cases which means less protected networks in case of node failures compered to the link failure case. This property gives us the motivation to overview the most common failure scenarios of operational networks in the upcoming section.

2.1.3 Failure Characterization

The authors of [39] spent 6 months with collecting IS-IS routing updates from a real ISP backbone network. The goal of their experiment was to classify the failures occur in pure IP networks, and to measure the time that IS-IS needs for completing the restoration process. They define two high-level categories: *planned* and *unplanned* outages where the former refers to the disruption caused by maintenance, and the second one includes all unexpected cases. Unplanned outages are further divided to *single link*, *router related* and *optical layer* failure sub-categories. Interestingly, the paper points out that the single link failures are responsible for 70% of the unplanned outages, while only 3% of the links cause 55% of the failures in overall. The study also reveals that for a set of links the frequency of the failures varies between 1 to 40 hours, and the recovery is in order of seconds.

Similarly to the previous report, Watson *et al.* [40] made experiments on a mid-size customer network, MichNet, to reveal the source and duration of instabilities in an OSPF network. They collected data from four geographically distributed probes over a one year period. One of their main findings was that the major source of instabilities are the *individually flapping links*.

Again, [41] pointed out that more than 80% of the failures are *transient*, lasting less than a couple of minutes. The shortest outages are mainly caused by overloaded routers that fail to process the heartbeat messages, and mistakenly consider the link to the neighboring router to be down. They found that the faulty optical equipment is responsible for the remaining part of the transient failures. Besides, the authors observed that 70% of the failures are isolated, caused by a single link outage.

However, it is common in real life that by cutting a single, high capacity cable brings multiple seemingly independent links down. The obvious reason lies in resource sharing: virtual connections are often used to connect logically independent networks. In other words, the logical representation of the network as seen by the IP control plane is often quite different from the physical layout. Consequently, a single link failure can be perceived as multiple independent link failures in the IP layer. Similar is the case of lightpaths in optical networks, Dense Wavelength-Division-Multiplexing wavelength channels, Generalized MPLS label switched paths, etc.; these low level connections may aggregate multiple IP links [42]. A convenient way to express the statistical dependency between a certain group of links or nodes that are expected to fail jointly is called *Shared Risk Groups* (SRGs). In the SRG model, if a link (or node) fails then all the links (nodes) that share an SRG with it will also fail, and correspondingly all traffic will be interrupted on these links (nodes). Throughout the Dissertation we take SRGs into considerations and incorporate this anomaly into our failure model.

As we can conclude the most usual reason for traffic disruption is caused by individually failing elements that can sometime manifest in simultaneous failure events. Usually, the set of malfunctioning devices is small and can be well-defined. We base our failure model on these statements throughout the Dissertation.

2.2 Notations and Methodology

2.2.1 Graph model

We model the network topology by a simple, weighted, symmetric directed graph $G(V, E)$ where V is the set of nodes and E is the set of directed arcs. We assume that the graph is 2-edge-connected, meaning that if we remove a single edge from the network, it still remains connected. Let $n = |V|$ and $m = |E|$, let $\overline{E}$ denote the complement arc set, let $\deg(v)$ denote the node degree of $v \in V$ and let $\text{neigh}(v)$ be the set of out-neighbors of v in G. We suppose that there is an IGP routing protocol running in the background that explores the graph and builds the shortest path trees. The link costs are represented by a cost function $c : E \mapsto \mathbb{Z}^+$. The cost of an edge (i, j) is denoted by $c(i, j)$. We assume that costs are symmetric: $c(i, j) = c(j, i)$. The distance between $x, y \in V$ nodes in the network is $\text{dist}(x, y)$. The networks do not include broadcast LANs and we also presume that there is no support for Equal-Cost MultiPath (ECMP). If shortest paths of equal cost are present, then ties get broken arbitrarily.

We also refine the definition of SRGs with cautious restraints. The reason is that the general SRG model can contain arbitrary set of network components, even ones residing in far-away portions of the network. Assuming all kinds of simultaneous failures in the network can make the easiest problem impossible to solve. A more realistic approach is to consider failing elements somehow correlated as it is described by the LFA standard. The *local SRGs* [8] is a sub-category of SRGs where a set of links is connected at one end of the same router. Local SRGs are not just frequent in practice, but also can be configured at each router locally and independently from each other that eliminates the need for a separate signaling protocol that would distribute SRG information throughout the network. For practical reasons, we allow only SRGs that do not cut the network into separate subgraphs, or formally the SRGs that leave the graph 1-connected in case of their removal from the network. Without having the restriction in place, SRGs could easily violate the 2-edge-connected graph model and would also bring unrealistic failure cases where nodes become instantly isolated in case of a single link failure that is usually not the case in operator networks.

2.2.2 The LFA definition

"The greatest ideas are the simplest", wrote William Golding in "Lord of the Flies". It absolutely applies to the Loop-Free Alternates. Briefly, we carefully select secondary next-hops for source-destination pairs that can be used in case of failures. Recall that the selection goes *proactively* and the reaction to the failure is *local*. Due to the latter requirement, inconsistent routing entries will be inevitably present in the network and our goal is to eliminate the possibility of creating a forwarding loop. This is satisfied if the shortest path of the LFA does not traverse the source node itself. Let

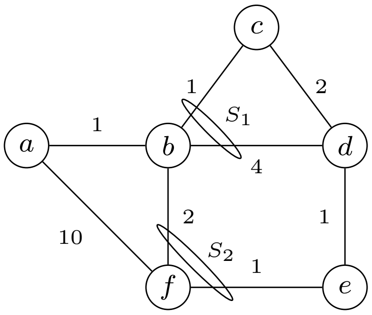


Figure 2.4: The sample network with local Shared Risk Groups S_1 and S_2, each containing two links

us overview these principles on the topology depicted on Fig. 2.4. The traffic from a to d flows on the default shortest path a-b-c-d, and we assume the failure of link (a, b). Due to the failure, the only remaining neighbor of a is f but the question if it is an LFA to d? The answer is yes, since the shortest path f-e-d does not include the source node, a.

By applying the previous notation, we get a formal definition of LFA. For some source s and destination d, let e be the default next-hop of s towards d. Then, some neighbor t of s is a *link-protecting LFA* [8] for s to d if

i) $t \neq e$, where e is the next-hop from s to d and

ii) $\text{dist}(t, d) < \text{dist}(t, s) + \text{dist}(s, d) \quad .$

Also, there is another popular terminology to phrase the same: we say that t is an $s \rightarrow d$ LFA if it is not *upstream* to s in the shortest path tree, rooted at d.

On the other hand, the presence of SRGs in our model makes this statement incomplete. Note on Fig. 2.4 that the local SRGs, S_1 and S_2, contain two links each. The interpretation is that if one of the links, say (b, c), fails then the other one, (b, d), is also expected to fail jointly. In our example d would be a perfect link-protecting $b \rightarrow c$ LFA, unless S_1 would not be part of the network. However, thanks to the common failure domain, if (b, c) fails then (b, d) goes down as well and b will no longer have direct connection to d. For this reason we shape our LFA definition for the case of SRGs, but first we introduce the formal description of local SRGs.

Let $S = \{(i, j) \in E\}$ be an SRG containing a set of links (i, j). The number of links is at most $\deg(i) - 1$, but for S to be a local SRG we require that for any two $(i, j) \in S$ and $(u, v) \in S$: $i = u$. Note that SRGs in our model can, and usually are, asymmetric, that is, $(i, j) \in S$ does not imply $(j, i) \in S$. Now, for each directed arc $(i, j) \in E$ we can create the union of SRGs that include (i, j):

$$S(i,j) = \bigcup_{S:(i,j)\in S} S \ .$$

With this in mind, we can continue by formulating the definition of LFA for the case of local SRGs. A candidate node t is an *SRG-disjoint link-protecting LFA* if the following applies:

Definition 1. *For some source s, destination d, and default $s - d$ next-hop e, a neighbor t of s is an SRG-disjoint link-protecting LFA for s to d if*

i) $t \neq e$,
ii) $\mathrm{dist}(t,d) < \mathrm{dist}(t,s) + \mathrm{dist}(s,d)$, *and*
iii) $(s,t) \notin S(s,e)$.

Similarly to link protection, there are LFAs that are able to protect against the failure of the next-hop. For instance, as we already showed on Fig. 2.4, node f is a link-protecting LFA for $a \to d$ since the shortest-path from f does not pass the source node, a. What is more, node f also protects against the failure of the next-hop b as well, since the $f \to d$ shortest path does not traverse b either. We call this property — extended with the SRG definition — the *SRG-disjoint node-protecting LFA* condition.

Definition 2. *For some source s, destination d, and default $s - d$ next-hop e, a neighbor t of s is an SRG-disjoint node-protecting LFA for s to d if*

i) $t \neq e$,
ii) $\mathrm{dist}(t,d) < \mathrm{dist}(t,s) + \mathrm{dist}(s,d)$,
iii) $\mathrm{dist}(t,d) < \mathrm{dist}(t,e) + \mathrm{dist}(e,d)$, *and*
iv) $(s,t) \notin S(s,e)$.

In addition, node-protection has a special condition for the case when $e = d$, that is, when d is the immediate next-hop of s. Since no LFA can protect against the failure of the destination node itself, in such cases s relaxes the pessimistic failure assumption and presumes that it is only the link (s, d) that failed and not d itself, and thus it can resort to a link-protecting LFA. This treatment of the *last-hop problem* is common in IPFRR [26].

What is left is to provide an adequate measure for the level of LFA protection of a given network. Simply we use the ratio of protected versus unprotected node pairs that is expressed as follows:

$$\eta_{\mathrm{LP}\,/\,\mathrm{NP}}(G) = \frac{\#\ (s,d) \text{ pairs with link/node-protecting LFA}}{\#\text{all } (s,d) \text{ pairs}} \ . \tag{2.1}$$

We call $\eta_{\mathrm{LP}}(G)$ and $\eta_{\mathrm{NP}}(G)$ link and node-protecting *LFA coverage* respectively. This value depends on the layout of the topology, as well as the link cost settings and finally the density of SRGs in the network. Our example on Fig. 2.4 holds $\eta_{\mathrm{LP}}(G) = 0.73$ and $\eta_{\mathrm{NP}}(G) = 0.66$.

The first goal of the Dissertation is *to raise the level of LFA coverage* by proposing smart changes to the topology. The way we approach this problem is outlined in the following section.

2.2.3 Methodology

The methodology of the research follows the scheme where theoretical results are obtained with analytical techniques and then get verified with numerical evaluations. In many cases the complexity of the problems is *nondeterministic polinomial (NP)* that requires to formalize Integral Linear Programs to find the optimal solution. Since LP solvers usually run with very poor performance, we propose *approximation algorithms* that perform close to the optimum. The set of *simulation* tools that we developed to support our research is publicly available on GitHub [43]. Most of the source files are written in C++ and they build on open source libraries like LEMON [44], Gurobi [45] or the BOOST unit test framework [46].

We use a wide variety of sample networks as inputs. The collapsed AS1221, AS1755, AS3257, AS3967 and AS6461 topologies are taken from the Rocketfuel dataset [47]. These graphs come from real service provider networks and inferred link costs. We also use the Abilene, Italy, NSF, Germany, AT&T and the extended German backbone (Germ_50) from [48]. Further topologies are obtained from the Topology-Zoo project's dataset [49] and we set costs randomly wherever link costs were not available. We have removed all parallel arcs and made links and costs symmetric.

2.3 New Results

In this section we take advantage of the major vulnerability of LFA that is the dependency between the layout of the topology and the level of LFA protection. We show how to *select and add complement edges* to the network that leads to high or even perfect protection.

2.3.1 Motivation

Latency sensitive media streams require ultra fast reaction to failures. The distributed scheme of IGP networking prevents to heal the network fast enough, therefore a local and proactive IPFRR scheme is necessary to be applied. As we have seen in Sec. 2.1.2, the only IPFRR method that was able to gain remarkable attention so far is **the Loop-Free Alternates that does not provide protection against all the failure cases**. On the other hand, since LFA selection is already available in major commercial router implementations [36, 37, 38], **there is a huge potential for the instant and seamless upgrade to a full IPFRR protected network** once we can get rid of this imperfect limitation of LFA. To improve the level of LFA protection in the network, we take advantage of the dependency between the number of protected node pairs and the layout of the topology. The idea is to add SRG-disjoint complement edges to the network that render available alternates to otherwise unprotected source-destination pairs.

The example on Fig. 2.5 briefly demonstrates this proposition. The node pair b-c is unprotected against the failure of link (b, c), since the only available neighbors of b, a and f, are upstream to the destination c. Now if we insert the link (b, e) with sufficiently high link cost setting that prevents the traffic to flow back to b, then e becomes an SRG-disjoint link-protecting $b \to c$ LFA. Recall that with this move we simply rely on the shortest path based delivery from e to c which is already the default behavior of routers. The only required step is to install the LFA in the forwarding rules of b that is far simpler than any other IPFRR proposal that would

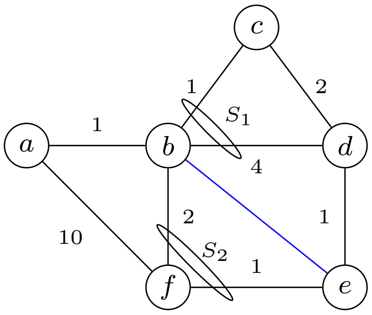


Figure 2.5: Extending the sample network with an additional link (marked with blue) for improving LFA coverage.

require additional signaling or the utilization of overridden protocol header fields to achieve the same goal.

Our goal is therefore to improve the LFA protection by inducing minimal changes to the topology. We achieve this task by augmenting the network with SRG-disjoint complement edges. During this process we seek answers for a couple of interesting questions. First and foremost, we must take into account that *default shortest paths must not be ruined* as they usually comply with carefully designed routing principles that follow special operator requirements. Second, we would like to know *if the problem is solvable at all for any given topology*, and if so, then *how to find the minimum set of complement edges that by adding to the network gives perfect protection.* Finally, it is also important to know *how to generate the set of complement edges that increases LFA coverage the most with a given upper limit on the number of new links that can be added to the network.*

2.3.2 Problem Formulation

Let us continue with the formal definition of the problem. Building on the definition of SRG-disjoint LFAs (Def. 1), we define the correlated failure tolerant LFA graph extension problem. Here, the task is to augment a weighted graph *with the minimum number of new links* with properly selected costs, so that LFA coverage becomes 100% and shortest paths remain in place. Formally:

Definition 3. Correlated failure tolerant LFA graph extension problem (minLFA$_{\mathrm{SRG}}$): *Given a simple, weighted, symmetric digraph* $G(V, E)$, *a set of SRGs* $\mathcal{S} = \{S\}$, *and an integer* l, *is there a symmetric arc set* $F \subseteq \overline{E}$ *with* $|F| \leq l$ *and properly chosen costs,* *so that* (i) *for the SRG-disjoint link-protecting LFA coverage* $\eta_{\text{LP}}(G(V, E \cup F)) = 1$ *and* (ii) *the shortest paths in* $G(V, E)$ *coincide with the shortest paths in* $G(V, E \cup F)$*?*

In order to satisfy the latter requirement, namely to keep shortest paths intact, we set link costs for the links added to the network to a sufficiently large value, say, larger than the length of the longest shortest path. In our model, the links are both-way directed so whenever we insert an arc its reverse is immediately added too, formally $(i, j) \in F \Rightarrow (j, i) \in F$. This case seems to be more relevant in practice, but we note that our model does not mandate this limitation in any way.

The above definition is straightforward to adapt to the the node-protecting case as well by substituting $\eta_{\text{LP}}(G)$ with $\eta_{\text{NP}}(G)$. Note that newly inserted links are not part of any existing SRGs and they never form new ones. This consideration comes from the fact that the role of the additional links is to provide ultimate reliability in the network without being affected by any other failure of network elements.

The problem of finding the minimum set of complement edges that brings LFA coverage to perfect is rather complex. In order to precisely define the level of "hardness" of this problem we must get familiar with two *complexity classes*: P ("easy") and NP ("difficult"). Problems belong to P can be solved in polynomial time, or we say that these are the problems that can be solved by today's regular computers (e.g. the problem of traversing any given graph [50]). On the other hand, the NP complexity class consists of decision problems that can be solved in non-deterministic polynomial time, which is in fact an unrealistic computational model. We assume that in case of a solution is provided for such a problem, then it can be verified in polynomial running time. For example finding a cycle in a graph that visits every vertex exactly once is in NP (Hamilton cycle [51]), but to verify if a given cycle is Hamiltonian or not goes in polynomial time. We call a problem NP-hard if it is at least as hard as any problem in NP. The intersection of NP and NP-hard defines a certain point in computational complexity that is called NP-complete. The method used to define a problem's complexity is called *reduction* that basically converts the given problem into another with known complexity [52].

The next result states that the minLFA$_{\text{SRG}}$ problem belongs to this "difficult" problem class that we called NP-hard above. This means that there is no known algorithmic technique to solve arbitrarily large problem instances in reasonable time. Formally:

Theorem 1. *The* minLFA$_{\mathrm{SRG}}$ *problem is NP-complete.*

The following definitions are required for the reduction used in the proof:

Definition 4. Bipartite graph: *The $G(A \cup B, C)$ graph is bipartite if the vertices can be decomposed into two disjoint sets, $A \cup B = V$, and for each $c(i,j) \in C$ edge either $i \in A$ and $j \in B$ or $j \in A$ and $i \in B$.*

Definition 5. Minimum set cover problem (minSC): *Given some positive integer p, is there a set of nodes $B^c \subseteq B$ with $|B^c| \leq p$, such that every node in A has a neighbor in B^c?*

Proof. The problem is in NP, as there exists an Integer Linear Program that can be formulated to solve minLFA$_{\mathrm{SRG}}$. By extending the definition of LFA to the case of SRGs (Def.1.), it makes the ILP proposed in [53] applicable to minLFA$_{\mathrm{SRG}}$. To show that it is NP-complete it is enough to find an NP-complete problem and convert its input G' to the input G of minLFA$_{\mathrm{SRG}}$ and verify that minLFA$_{\mathrm{SRG}}$ is solvable if and only if the referenced problem solves G'. A very similar problem has been proven to be NP-complete [53] by reducing graph extension to the minimum set cover problem in bipartite graphs (Def. 5).

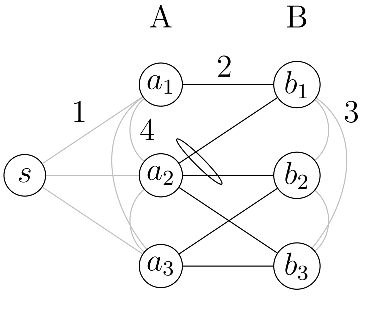


Figure 2.6: Illustration for converting the bipartite graph, $G'(A \cup B, C)$, to $G(V, E)$.

The idea is to integrate G' into G and show that achieving perfect LFA coverage is only possible with satisfying *minSC* too. On Fig. 2.6 we extend the bipartite graph with node s, we create internal, SRG-disjoint, linkage within A and B (gray) and we apply specific link cost settings. Now there are three possible scenarios for having a destination: i) d=s, ii) d=a_i, iii) d=b_i. In i) each node have an LFA to d since there always exists an LFA from every $a_i \in A$ through the internal links or similarly, every $b \in B$ has an SRG disjoint alternate either in A or B.

On the other hand, if d resides in A (ii), then s do not have an LFA to d since all its neighbors in A are upstream, therefore the only way to render an $s \to d$ LFA is by adding a complement edge of high cost, (s, b_j), where b_j is neighbor of d. Recall that by definition the complement edge is SRG-disjoint from (s, d).

Lastly, in case of iii) all $b_i \in B \setminus \{d\}$ have an LFA to d thanks to the internal links and so does every $a_i \in A$ can access d through an SRG-disjoint (a_i, b_i) link and the corresponding internal connection, (b_i, d). The only node that is left is s, but one of the previously added complement edges due to ii) must provide an LFA to the destination in B too. What is left is to verify that $B^c \subseteq B$ selected by complement edges is indeed a minimum set cover. Nodes in B^c must be adjacent to each node in A, otherwise there would be unprotected $s - a$ pairs which is not the case as per ii). This concludes the proof. □

Asking for link extension that attains 100% failure coverage is often too ambitious goal. Therefore, we also consider a relaxed version of the problem, called the correlated failure tolerant *LFA graph improvement* problem, which asks for realizing the highest improvement possible by adding only a limited number of new links.

Definition 6. Correlated failure tolerant LFA graph improvement problem: *Given a simple, weighted, symmetric digraph $G(V, E)$, a set of SRGs $\mathcal{S} = \{S\}$, and two integers l and k, is there a symmetric arc set $F \subseteq \overline{E}$ with $|F| \leq l$ and properly chosen costs, so that* (i) *at least k source-destination pairs have an SRG-disjoint link-protecting LFA in $(G(V, E \cup F))$ and* (ii) *the shortest paths in $G(V, E)$ coincide with the shortest paths in $G(V, E \cup F)$?*

The LFA graph improvement problem is also NP-complete, because otherwise we could solve $\text{minLFA}_{\text{SRG}}$ with solving LFA graph improvement with setting $k = |V^2| = n(n-1)$. Next, before jumping to the solution of the problem we examine if it is solvable at all for any arbitrary input.

2.3.3 The Solvability of $\text{minLFA}_{\text{SRG}}$

LFA is based on the idea to pass packets *to a neighbor that is not upstream* towards the destination. Unfortunately, such a neighbor often does not exist. The goal of $\text{minLFA}_{\text{SRG}}$ is to add complement edges between the source nodes of unprotected

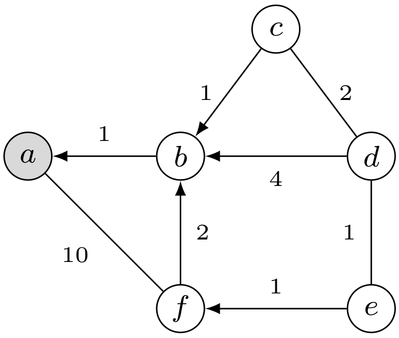


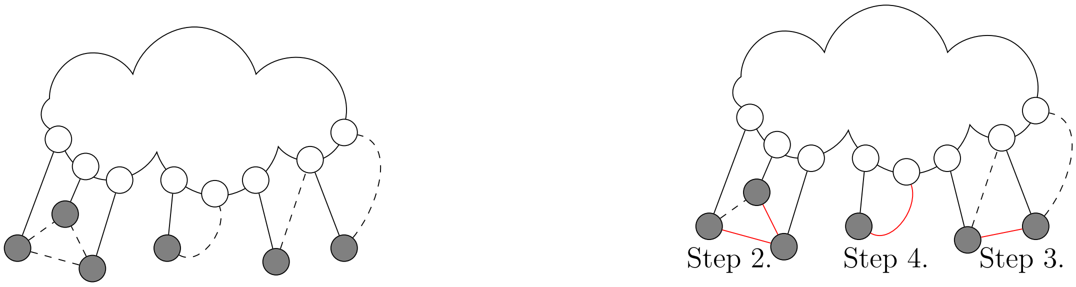
Figure 2.7: A sample graph where pre-processing is required to solve $\text{minLFA}_{\text{SRG}}$, otherwise node pair b-a remains unprotected. Arrows mark the shortest-paths tree rooted at a.

node pairs and the nodes that have alternate paths to the destination. However, if there is a single node that is used by *all the other nodes to reach a certain destination*, this task becomes impossible. We show this example in Fig 2.7. Here, all traffic destined to a enters a single last-hop router, b. In case of the failure of link (b, a), b is not able to reach a anymore, and what is worse, we cannot create an LFA by adding a new link because all nodes of the graph use b to reach a. This inevitably prevents to solve $\text{minLFA}_{\text{SRG}}$. The authors of [53] made this observation explicit.

Proposition 1. *Let $G(V, E)$ be a simple, weighted graph. Now, there is some integer l so that the LFA graph extension problem is solvable in G for l, if and only if each $d \in V$ has at least two children in the d-rooted shortest path tree.*

Accordingly, by decreasing the cost of link (f, a) from 10 to lower than $\text{dist}(f, a) - \epsilon$, say 2.9, then f becomes a child of a that instantly makes the graph solvable. The authors suggest a rudimentary strategy that either inserts or re-uses existing links in the network, but the resulting changes to the topology are far from minimal. Therefore, we introduce the *pre-processing* problem that aims to make $\text{minLFA}_{\text{SRG}}$ solvable by making minimal changes so that all nodes satisfy Prop. 1:

Definition 7. LFA graph extension pre-processing problem: *Given a simple, weighted, symmetric digraph $G(V, E)$ containing S set of SRGs, find a modified graph $G'(V, E')$ and a modified weight set, so that the* $\text{minLFA}_{\text{SRG}}$ *is* solvable over G'. *Is there an E' with the minimum set of new and altered edges that results in a minimum difference of shortest paths in G and G'?*



(a) Step 1. of pre-processing

(b) Step 2-3-4-5 of pre-processing

Figure 2.8: A sample graph that requires pre-processing. Gray nodes mark $v_p \in V_p$ Solid black and red lines mark the last hop of the shortest path rooted at nodes $v_p \in V_p$. Dashed lines represent the links not used for packet delivery. Red links are the output of the improved pre-processing algorithm.

Our task is to design an algorithm that converts G to G' by inducing minimal changes to the topology. The idea is to obtain the minimal set of unused links and to adjust their cost so that each node in V eventually gain two children. Rather than solving the problem directly on G, we introduce an intermediate step, a subgraph of G, that makes possible to apply graph optimization algorithms on a reduced size of input. Let $G_p(V_p, E_p)$ denote the subgraph extracted from $G(V, E)$ so that none of $v_p \in V_p$ satisfies Prop. 1 (nodes reached by one shortest path only), and $E_p \in E$ where both ends are in V_p.

First, we seek for the smallest set of edges in E_p that covers each connected node in V_p (*minimum edge cover*), then we pair the remaining nodes arbitrarily and we assign the link costs so that only shortest paths within G_p alter:

Algorithm 1 Improved pre-processing algorithm

```
    Input: $G(V, E)$
    Output $G'(V', E')$

  1.) Select $G_p(V_p, E_p) \subseteq G(V, E)$.
  2.) Find minimum edge cover in $G_p(V_p, E_p)$.
  3.) Pair remaining nodes of $V_p$ and insert edges between them.
  4.) Insert new edge between $u, v \in V_p$ where $u$ is the last not pre-processed node.
  5.) Assign cost to the links selected above so that only shortest paths between
      $v \in V_p$ alter.
```

We demonstrate these steps on the example depicted in Fig. 2.8. The subgraph

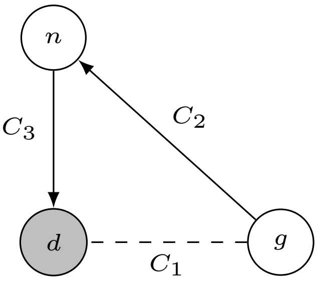


Figure 2.9: Sample network for the proof of Corollary 1

.

$G_p(V_p, E_p)$ consists a triangle of 3 nodes and 3 individual nodes (marked with dark gray). Links excluded from packet delivery (i.e. not part of any shortest paths) are represented by dashed lines, and conversely we denote with solid black lines the links carrying traffic. The goal of the algorithm is to create at least two "entries" for each $v_p \in V_p$ in minimal steps that simply translates into assigning two solid lines to each dark grayed node in the example. Common sense suggests that the most effective way to achieve this is by pairing the nodes of V_p, because then two new children are created in on step. Accordingly, minimum edge cover selects two links in the left side triangle, then by inserting a new link we pair two nodes, and finally we pick an unused link for the last remaining node of V_p (red lines in Fig. 2.8b). What is left is to assign sufficiently low link costs so that each $v_p \in V_p$ gains exactly one new child.

We have not verified yet that Alg. 1 indeed makes the graph solvable as per Prop. 1.

Corollary 1. *The algorithm presented in Alg. 1 guarantees that all $v \in V$ nodes have at least two children in the v-rooted shortest path tree.*

Proof. Since the graph is 2-connected, there are at least two incident edges at each vertex. Next, we show that by adjusting the cost of an unused link creates an additional child for any given $v_p \in V_p$. Node d has a single child on Fig. 2.9 and we claim that there exists a link cost setting, C_1, of the unused link that turns g into a second children of d without modifying n. To make the g-d distance shorter via the direct link compared to $\text{dist}(g, d)$ the following is required: $C_1 < C_2 + C_3$. On the other hand, we must also ensure that the shortest path, n-d, does not change:

$C_3 < C_2 + C_1$. Arranging the inequalities we get:

$$C_3 - C_2 < C_1 < C_2 + C_3,\ C_1 > 0 \tag{2.2}$$

Adjusting C_1 accordingly is always possible since $C_1, C_2, C_3 \in \mathbb{N}$. Note that the same holds if C_2 is replaced with $\text{dist}(n, g)$. □

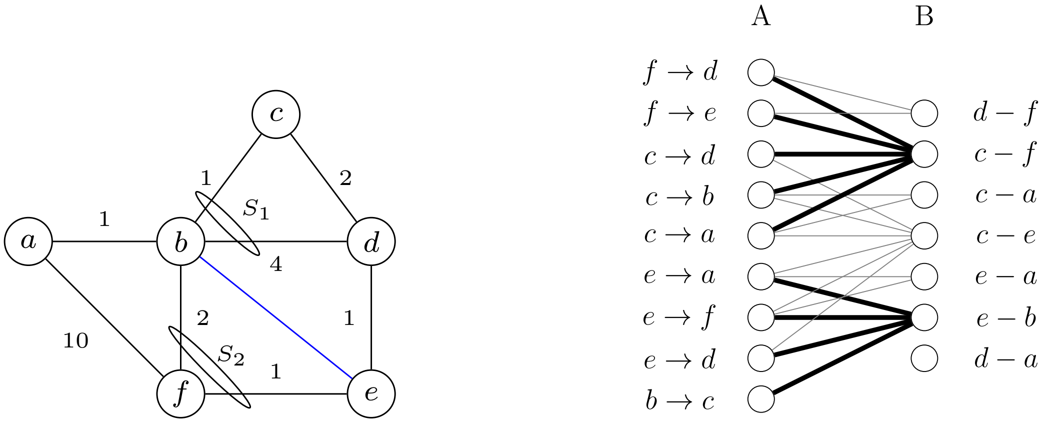
The algorithm we propose outperforms the pre-processing used by [53] in Fig. 2.8b as it only requires 2 new additional edges and solves the problem with the modification of 4 edges in total. The compared algorithm carries out the same task by inserting 5 new links that is a 25% gain on the number of touched edges for Alg. 1. What is left is to characterize the running time of Alg. 1.

Theorem 2. *The algorithm presented in Alg. 1 has polynomial running time and terminates with optimal result.*

Proof. Subtracting G_p from G takes $O(n^2)$ steps where the computation is dominated by the Dijkstra algorithm. The check whether a node has a single entry can be integrated into the Dijkstra (flag if there is a single next-hop towards all the destinations). The minimum edge cover on G_p seeks first for maximal matching in the graph where the task is to find as many edges as possible so that each vertex is adjacent only to a single edge. The set is then greedily augmented by the edges that connect the remaining disconnected nodes. Maximal matching takes $O(n^2 m)$ steps and the greedy augmentation adds another $O(m)$ toll on the computation where m is the number of edges. Making pairs of the isolated nodes of G_p requires no more than $O(n)$ steps, as one only need to iterate over the set of nodes and connect each one to the previous. As a result we can conclude that Alg.1 has a polynomial complexity of $O(n^2)$ that completes the proof. □

2.3.4 Solving $\text{minLFA}_{\text{SRG}}$ with the Bipartite Graph Model

The first step to solving the $\text{minLFA}_{\text{SRG}}$ problem is to build a suitable model. As the proof of Theorem 1 suggests the problem can be reduced to finding the minimum set cover in bipartite graphs [50]. The idea is therefore to build a construction, a suitable bipartite graph model, which then can be used to solve $\text{minLFA}_{\text{SRG}}$ as well.



(a) Sample graph (b) Bipartite graph

Figure 2.10: Sample bipartite graph representations for the SRG-disjoint link-protecting case. Nodes in A represent the unprotected node pairs while nodes in B mark the complement edges.

But how to construct such a model? Let us return to our explanatory topology on Fig. 2.10a. Previously, we found that the graph does not have full link-protecting LFA coverage. This can be improved by installing the link (b, e) that renders an LFA for the unprotected node pair, $b \to c$. On the other hand, there exists a complement arc in the network, (d, a), that do not raise the level of protection at all. Therefore our task is to find the smallest symmetric subset of the complement arc set $\overline{E}$ so that each unprotected source-destination pair gets an LFA.

Let $(s_i, d_i) : i \in 1, \dots, k$ be the set of unprotected source-destination pairs and let $\{(u_j, v_j) : j \in 1, \dots, l\}$ be the set of complement arcs $\overline{E}$ from which reverse arcs were eliminated, i.e., the set contains either (u_j, v_j) or (v_j, u_j), but not both. Let $G'(A, B, F)$ be an undirected bipartite graph with node set $A \cup B$ and edge set F, where we add a node $a_i \in A$ corresponding to each unprotected $(s_i, d_i) : i \in 1, \dots, k$ and a node $b_j \in B$ for each arc $(u_j, v_j) : j \in 1, \dots, l$, and we connect some $a_i \in A$ to some $b_j \in B$ in G' if and only if arc (u_j, v_j) or (v_j, u_j), when added with suitably large cost to G, would create an SRG-disjoint link-protecting LFA to (s_i, d_i). This amounts to checking whether Definition 1 would hold for (s_i, d_i) on the graph augmented with (u_j, v_j) and (v_j, u_j).

The bipartite graph $G'(A, B, F)$ has $O(n^2)$ nodes and $O(n^4)$ arcs, and it can be built in $O(n^2(n^2 \log n + nm))$ time as we need to perform an all-pairs-shortest path calculation for each of the $O(n^2)$ complement arcs. Furthermore, the operation of adding a link (u_j, v_j) to G corresponds in G' to deleting the node b_j and all its neighbors from A. Since we take care of leaving the shortest paths in G intact, the resultant bipartite graph remains a valid representation of the LFA graph extension problem on the augmented graph. What is left is to solve the minimum set cover problem ($minSC$) over G'.

Fig. 2.10b shows the constructed bipartite graph model for the input topology of Fig. 2.10a. We marked the minimum cover with thick lines so it is easy to see that by adding only 2 complement arcs, (c, f) and (e, b) (and their reverse arcs), the LFA coverage becomes 100% in the link-protecting case. Interestingly, the node with the highest degree from B, $(c - e)$, is not part of the given cover implying that there are strategies performing better than a simple greedy selection.

Theorem 3. *There is an algorithm to build the bipartite representation for any instance of the node-protecting* minLFA$_{\text{SRG}}$ *problem in $O(n^3)$ steps.*

Proof. We add a node to A corresponding to each (s, d) that has no SRG-disjoint node-protecting LFA. Additionally, we add a node to B for each complement arc in $\overline{E}$ eliminating the reverse arcs. For some $a_i \in A$ and some $b_j \in B$ we add an (a_i, b_j) arc to F if for the corresponding source-destination pair (s_i, d_i), complement arc (u_j, v_j) and next-hop n of s_i towards d_i, one of the below conditions holds. First, in case of the *last-hop problem*, we settle with satisfying the link-protecting condition only:

- $n = d_i$, $u_j = s_i$ and $\text{dist}(v_j, d_i) < \text{dist}(v_j, s_i) + \text{dist}(s_i, d_i)$; or
- $n = d_i$, $v_j = s_i$ and $\text{dist}(u_j, d_i) < \text{dist}(u_j, s_i) + \text{dist}(s_i, d_i)$;

Otherwise, when the destination is not adjacent to the source, the neighbor must also fulfill the node-protecting requirement:

- $n \neq d_i$, $u_j = s_i$, $v_j \neq n$, $\text{dist}(v_j, d_i) < \text{dist}(v_j, s_i) + \text{dist}(s_i, d_i)$ and $\text{dist}(v_j, d_i) < \text{dist}(v_j, n) + \text{dist}(n, d_i)$; or
- $n \neq d_i$, $v_j = s_i$, $u_j \neq n$, $\text{dist}(u_j, d_i) < \text{dist}(u_j, s_i) + \text{dist}(s_i, d_i)$ and $\text{dist}(u_j, d_i) < \text{dist}(u_j, n) + \text{dist}(n, d_i)$.

This can be done in $O(n^3)$ time, checking the above conditions for each of the $O(n)$ neighbors for each $O(n^2)$ node in B. □

A further attractive aspect of this model is that the construction of the bipartite graph is indifferent to whether local SRGs are present in the network or not. Therefore it also provides solution for the correlated failure free case, $\mathcal{S} = \emptyset$. Since the only condition we need to replace is Definition 1, then neither the size of the bipartite graph nor its construction time gets affected by the change.

2.3.5 Algorithms

We have found in Sec. 2.3.2 that the minLFA$_{\text{SRG}}$ problem is NP-complete. This level of complexity makes uncertain to obtain the optimal results in reasonable time for larger networks. On the other hand, we recognize that finding minimum set cover in a bipartite graph is equal to the problem of finding minimum vertex cover in hypergraphs [54] for which several efficient heuristics are available from the literature. In particular, we discuss the Lovasz-Johnson-Chvatal algorithm from [55], the SBT algorithm from [56], the RSBT, MSBT algorithms from [54]. An added benefit of using these heuristics is that they immediately provide solutions to the LFA improvement problem beside LFA graph extension; we only need to terminate the algorithms after a pre-defined number of iterations, i.e., after adding a pre-defined number of new links.

The Lovász-Johnson-Chvatal (LJC) method

The algorithm of Lovász-Johnson-Chvatal (LJC, [55] adds the highest degree node $v \in B$ to the cover B^c, v and its neighbors in A are deleted from G' and the algorithm proceeds to the next iteration.

Lovász shows that the size of the cover provided by this algorithm is within a logarithmic factor of the optimum: $t_{opt} \leq t_{LJC} \leq t_{opt} * (1 + \log_2 |B|)$ where t_{opt} is the cardinality of the optimal cover and t_{LJC} denotes the cardinality of the cover B^c from LJC [55]. This means that the algorithm solves the LFA graph extension problem by adding only logarithmically more new links than necessary.

The greedy algorithm is remarkably fast. Unfortunately, it is not guaranteed that the cover returned by the LJC algorithm is *minimal in the sense of inclusion*, which basically means that some proper subset of the solution would also be an adequate cover. This might render the LJC algorithm hugely impractical in certain cases.

Algorithm 2 LJC

```
$B^c \leftarrow \emptyset$
while $A \neq \emptyset$ do
    $v \leftarrow \operatorname{argmax}_{b \in B} \deg(b)$
    $B^c \leftarrow B^c \cup \{v\}$
    $A \leftarrow A \setminus \operatorname{neigh}(v)$
    $B \leftarrow B \setminus \{v\}$
end while
```

Algorithm 3 SBT

```
1: $B^c \leftarrow \emptyset$
2: while $A \neq \emptyset$ do
3:     $v \leftarrow \operatorname{argmin}_{b \in B} \deg(b)$
4:     if $\exists n \in \operatorname{neigh}(v)$ with $\deg(n) = 1$
5:         $B^c \leftarrow B^c \cup \{v\}$
6:         $A \leftarrow A \setminus \operatorname{neigh}(v)$
7:     end if
8:     $B \leftarrow B \setminus \{v\}$
9: end while
```

Figure 2.11: Pseudo-codes of the LJC and SBT algorithms on graph $G'(A, B, F)$

The SBT algorithm

SBT was proposed in [56] to find an approximate cover that is, in contrast to LJC, minimal in the sense of inclusion. SBT seeks for the node $v \in B$ with the smallest degree and removes it from B. Additionally, if neigh(v) contains a node a that is covered by v only, then v is added to B^c as otherwise we could not cover A. In this case, we consider all v's neighbors as covered, remove them from A and proceed to the next iteration.

The RSBT algorithm

The Reverse SBT algorithm [54], as the name says, does the reverse of SBT in that in every iteration it chooses the node with the highest degree instead of the smallest degree. Consequently, the pseudo-code is the same as given in Algorithm 3 with the slight modification that instead of line 3 we write $v \leftarrow \operatorname{argmax}_{b \in B} \deg(b)$.

The MSBT algorithm

The Modified SBT algorithm [54] applies a small optimization step to SBT. Similarly to the SBT algorithm, in each iteration we choose the node $v \in B$ with the smallest degree and, if there are nodes in A covered only by v, we add v to B^c . If, on the other hand, $B \setminus \{v\}$ remains a cover, then we search for all the nodes $a \in A$ that are covered by exactly two nodes in B: v plus some other node, say, $w \neq v$, we add these ws to B^c and we remove them from B and all their neighbors from A.

Algorithm 4 RSBT

```
$B^c \leftarrow \emptyset$
while $A \neq \emptyset$ do
    $v \leftarrow \text{argmax}_{b \in B} \deg(b)$
    if $\exists n \in \text{neigh}(v)$ with $\deg(n) = 1$
        $B^c \leftarrow B^c \cup \{v\}$
        $A \leftarrow A \setminus \text{neigh}(v)$
    end if
    $B \leftarrow B \setminus \{v\}$
end while
```

Algorithm 5 MSBT

```
$B^c \leftarrow \emptyset$
while $A \neq \emptyset$
    $v \leftarrow \text{argmin}_{b \in B} \deg(b)$
    if $\exists n \in \text{neigh}(v)$ with $\deg(n) = 1$
        $B^c \leftarrow B^c \cup \{v\}$
        $A \leftarrow A \setminus \text{neigh}(v)$
    else
        for each $a \in \text{neigh}(v)$
        with $\deg(a) = 2$
            $w \leftarrow u \in \text{neigh}(a) \setminus \{v\}$
            $B^c \leftarrow B^c \cup \{w\}$
            $A \leftarrow A \setminus \text{neigh}(w)$
        end for
    end if
    $B \leftarrow B \setminus \{v\}$
end while
```

Figure 2.12: Pseudo-codes of the RSBT and MSBT algorithms on graph $G'(A, B, F)$

Note that the SBT, RSBT and MSBT algorithms generate covers that are minimal in the sense of inclusion.

2.3.6 Numerical Evaluations

The main task we undertook in our numerical studies was to determine which of the above heuristics works best for minLFA$_{\text{SRG}}$. In particular, we were curious as to how many new links are needed to achieve full LFA protection with the different algorithms both for the link-protecting and the node-protecting cases. We executed our measurements under different failure models. We start our discussion by setting $\mathcal{S} = \emptyset$, i.e. no SRGs are present in the network for having a proper baseline. Then we continue by incrementally increasing the number of SRGs for the measurements.

The topologies were chosen so as to ensure that the ILP proposed in [53] still runs — at least in the non-SRG case — and so we can compare the performance of the heuristics to each other as well as to the optimum. Before actually running the algorithms, the improved pre-processing algorithm was executed in order to ensure that the optimization problems were always solvable.

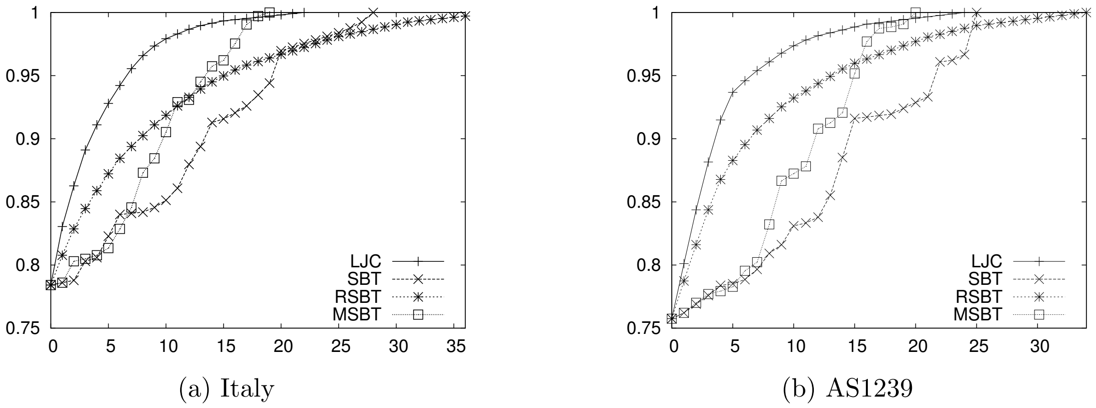
Table 2.1: Results of link-, and node-protecting LFA graph extension: topology name, number of nodes (n) and arcs (m); number of link costs and arcs added in the pre-processing phase ("Pre. c/e"), initial LFA coverage (η_0), number of new arcs in the optimal solution (ILP), and the number of added arcs ("ext") and execution time in seconds for each algorithm

							LJC		SBT		RSBT		MSBT	
	Topology	n	m	Pre. c/e	η_0	ILP	ext	time	ext	time	ext	time	ext	time
Link-protecting	AS1239	30	69	0/0	0.874	6	6	0.01	7	0.43	11	0.08	6	0.47
	AS1755	18	33	0/0	0.873	7	7	0.01	9	0.03	12	0.01	7	0.02
	AS3967	21	36	0/0	0.786	8	11	0.01	10	0.09	16	0.03	9	0.09
	AT&T	22	38	0/0	0.822	10	12	0.01	12	0.10	12	0.02	11	0.10
	Digex	31	35	0/0	0.316	22	27	0.80	27	2.02	46	1.80	27	1.74
	Germ_50	50	88	0/0	0.900	18	21	0.04	29	4.80	44	1.53	25	4.32
	Germany	17	25	0/0	0.694	9	12	0.01	12	0.04	13	0.01	11	0.03
	Italy	33	56	0/0	0.784	17	22	0.03	28	1.04	39	0.43	19	0.89
	NSF	26	43	0/0	0.860	11	12	0.01	13	0.18	28	0.09	13	0.16
	Carnet	44	43	34/34	0.818	16	19	0.04	16	4.33	16	0.25	16	4.21
	Bestel	84	93	11/11	0.343	68	91	5.57	82	378	128	276	75	312
	Deltacom	113	183	11/10	0.614	80	100	9.22	94	1222	131	490	91	989
	Average:	40.7	61.8	4.6/4.5	0.723	22.6	28.3	1.31	28.25	134.5	41.3	64.2	25.8	109.5
Node-protecting	AS1239	30	69	0/0	0.757	19	24	0.02	25	0.60	34	0.22	20	0.50
	AS1755	18	33	0/0	0.765	16	19	0.00	18	0.03	27	0.01	18	0.02
	AS3967	21	36	0/0	0.643	17	19	0.01	20	0.12	32	0.08	20	0.11
	AT&T	22	38	0/0	0.580	38	43	0.01	43	0.12	54	0.08	40	0.10
	Digex	31	35	0/0	0.312	29	36	0.98	39	1.72	50	1.63	34	1.55
	Germ_50	50	88	0/0	0.828	34	44	0.13	57	6.35	81	3.41	50	5.75
	Germany	17	25	0/0	0.562	18	22	0.00	22	0.04	27	0.03	19	0.03
	Italy	33	56	0/0	0.570	35	43	0.08	48	1.49	60	0.94	38	1.22
	NSF	26	43	0/0	0.634	18	24	0.02	34	0.41	38	0.26	23	0.33
	Carnet	44	43	34/34	0.746	100	102	0.15	101	2.68	107	0.74	100	2.07
	Bestel	84	93	11/11	0.312	106	137	7.78	134	351	173	274	124	262
	Deltacom	113	183	11/10	0.527	171	212	20.7	214	1220	255	599	197	890
	Average:	40.7	61.8	4.6/4.5	0.603	50.1	60.4	2.5	62.9	132	78.1	73.3	56.9	96.9

Results in networks without SRGs

As a subset of the minLFA$_{\text{SRG}}$ problem we assume single link and single node failures in the first round. The number of new links added and the running time by the different algorithms for both link-, and node-protecting LFAs are given in Table 2.1. The tables also report on the average number of new links added by the different algorithms.

The most important observations are as follows. First, the initial LFA coverage is usually about 70-90% in the link-protecting case and only 55-75% in the node-protecting case. This is expected, as node-protection is a stricter requirement than link-protection. Second, on most small and middle-sized networks adding only about



(a) Italy

(b) AS1239

Figure 2.13: LFA coverage in each iteration of different heuristics in the link-protecting case for Italy, and node-protecting case for the AS1239 topology.

a dozen or less new links is often enough to achieve 100% link-protection. This marks the huge potential to LFA-based network optimization. For node-protection, however, significantly more new links are needed, to the point that in larger topologies we need to virtually double or triple the number of links. Third, all heuristics perform surprisingly well, only overshooting the optimum by at most 5-15% in most cases and even finding the optimum for some networks. The MSBT algorithm is the clear winner both for link- and node-protection, with the SBT and LJC algorithms also working reasonably, while RSBT is the worst performer.

It seems that for larger networks, and especially in the node-protecting case, we need dozens of new links to achieve 100% LFA protection. This is clearly out of scope for most operators. Instead of aspiring to 100% protection, the LFA graph improvement problem therefore aims towards the more realistic goal of boosting the LFA coverage by adding only a small number of new links. Thusly, we also examined how the LFA coverage increases with each added new link in the subsequent iterations of the algorithms. The results for some selected topologies are depicted in Fig. 2.13. The most important observation is that **while MSBT is the most efficient in attaining 100% coverage with the smallest number new links, it is the LJC algorithm, by nature, that improves the LFA coverage the most in the initial steps**. The RSBT algorithm also performs well in this regard. With LJC, about 10-15% improvement in the LFA coverage can be realized by adding only at most 5 new links and another 10% with the next 5 links, putting the coverage in the 90-95% range, which may be enough in many practical scenarios.

Results in networks with SRGs

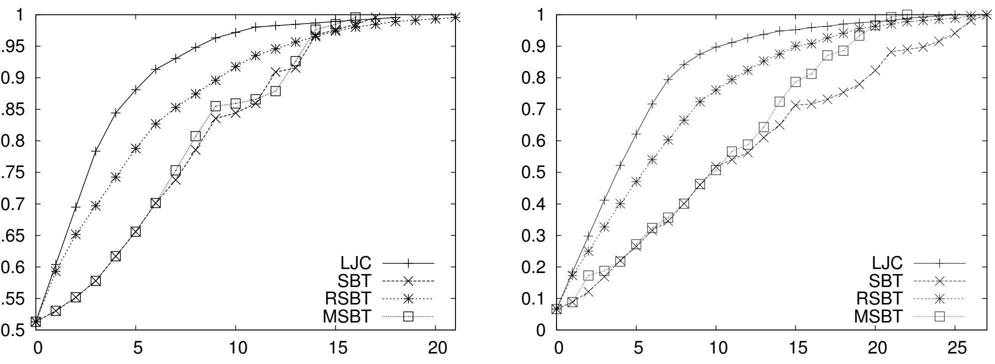
In the second round, we have extended our investigation to the case when certain sets of links are configured into local SRGs. Local SRG sets were generated according to an SRG-density parameter $\delta \in [0, 1]$, denoting the fraction of all possible adjacent dual-link sets to be selected as local SRGs. For $\delta = 0$ we add no SRGs at all (i.e., this case corresponds to the single failure scenario), and for the settings $\delta = 0.1$ ($\delta = 0.5$, $\delta = 0.9$, respectively), we add every adjacent link pair with probability 0.1 (respectively 0.5 and 0.9) as an SRG. Since δ is in fact only a statistical expectation on the density of SRGs, we generated 10 different sets of SRGs to each network and we executed the algorithms on all these scenarios, eventually reporting the average of the results.

The results themselves for some selected topologies are in Table 2.2 for both the link, and node-protecting cases under different choices of the SRG density δ. As a consequence of the complexity of the ILP, we have omitted calculating the optimal results for each SRG set. Our observations are as follows. First, we see that as SRG density increases the initial LFA coverage drops drastically. When the SRG density is only 10% the initial LFA coverage lags behind the single-failure case by only a mere 2-6% in both the link- and the node-protecting cases. However, for $\delta = 0.5$ the difference increases to about 20-25% for the better protected networks and about 10% for the less protected ones, to the point that only about half of the source-destination pairs can be protected by an LFA. Finally, when the SRG density is set to $\delta = 0.9$ the initial link-protecting LFA coverage falls to about 10-20%, and a bare 3-12% for the node-protecting case.

Surprisingly, we find that the number of new links to be added does not grow at a similar pace: in the link-protecting case for $\delta = 0.1$ we need about 2-3 more links than in the non-SRG case, while $\delta = 0.5$ introduces about 3-4 more links and even in the case of very large SRG density $\delta = 0.9$ we only need about 7-12 links more (depending on the network size) than when we do not have SRGs at all. This essentially means that even when 90% of every possible adjacent link pair is configured as an SRG we can still improve link-protecting LFA coverage to 100% with only a dozen or so new links. A possible cause behind the insensitivity of our results to SRG density might

Table 2.2: Link-, and node-protecting LFA graph extension results for some select topologies with SRGs: topology name; number of link costs and arcs added in the preprocessing phase ("Pre. c/e"); initial LFA coverage (η_0); and the average of number of added arcs ("ext") for each algorithm for small ($\delta = 0.1$), medium ($\delta = 0.5$), and large ($\delta = 0.9$) SRG density

			$\delta = 0.1$					$\delta = 0.5$					$\delta = 0.9$				
	Topology	Pre. c/e	η_0	LJC	SBT	RSBT	MSBT	η_0	LJC	SBT	RSBT	MSBT	η_0	LJC	SBT	RSBT	MSBT
Link-protecting	AS1239	1/0	0.85	8.3	8.4	13.4	7.8	0.66	15.1	15.7	24.6	14.6	0.21	21.5	20.4	32.8	17.8
	AS1755	0/0	0.84	7.9	9.2	12.1	7.8	0.60	12.1	12.9	15.0	11.5	0.15	15.1	12.9	17.7	12.1
	AS3967	0/0	0.75	11.1	10.9	17.0	9.6	0.51	12.9	13.3	21.0	12.0	0.16	15.2	13.7	24.2	13.5
	AT&T	1/1	0.77	13.1	12.8	13.9	12.5	0.50	17.2	15.0	19.8	14.6	0.10	19.3	16.2	22.6	15.9
	Digex	0/0	0.27	26.9	27.4	45.9	25.9	0.17	28.4	28.2	44.8	26.1	0.03	28.3	26.3	42.9	25.4
	Germ_50	0/0	0.85	25.4	35.7	49.5	27.6	0.59	33.0	40.4	64.1	35.9	0.14	38.6	35.2	69.0	34.8
	Germany	0/0	0.63	13.5	12.0	14.2	11.5	0.41	14.4	13.2	14.7	12.6	0.09	16.1	13.9	16.1	13.0
	Italy	1/1	0.74	22.4	27.9	39.2	20.9	0.49	27.1	30.1	43.2	25.6	0.11	32.1	28.7	45.1	27.3
	NSF	0/0	0.81	13.5	16.1	29.0	13.0	0.50	18.1	20.7	32.5	16.9	0.11	21.3	20.0	34.5	19.3
	Average:		0.72	15.8	17.8	26.0	15.2	0.493	19.9	20.9	31.0	18.8	0.124	23.1	21.0	33.8	20.0
Node-protecting	AS1239	1/0	0.73	23.6	25.7	38.3	21.2	0.57	29.4	33.6	45.4	27.9	0.17	34.8	35.5	51.1	33.8
	AS1755	0/0	0.73	18.9	18.4	27.0	17.7	0.49	22.7	22.8	28.4	20.9	0.12	26.6	25.6	30.8	24.1
	AS3967	0/0	0.60	19.4	21.0	31.7	19.3	0.40	21.6	24.6	31.5	21.9	0.12	24.5	26.4	32.4	25.1
	AT&T	1/1	0.54	43.5	43.7	53.5	40.0	0.31	44.8	46.2	52.1	40.9	0.07	44.1	44.4	47.2	39.3
	Digex	0/0	0.27	37.2	38.8	50.3	34.6	0.17	38.2	39.9	52.2	37.5	0.04	39.4	39.3	49.0	38.9
	Germ_50	0/0	0.77	44.9	62.6	81.7	49.1	0.51	51.0	64.4	85.1	56.2	0.11	56.8	56.5	88.3	54.6
	Germany	0/0	0.51	21.5	22.2	26.6	19.1	0.29	23.2	23.9	26.7	20.3	0.05	25.2	25.0	26.3	22.3
	Italy	1/1	0.53	42.3	48.5	62.1	39.4	0.35	45.2	47.5	61.6	42.8	0.08	47.2	45.4	63.2	45.2
	NSF	0/0	0.57	25.4	31.3	38.2	23.4	0.33	28.9	30.4	40.6	26.7	0.07	32.0	30.7	39.9	28.4
	Average:		0.584	30.9	34.8	45.1	29.4	0.381	34.2	37.3	46.8	33.2	0.091	37.6	37.2	47.8	35.3



(a) AT&T

(b) Germany

Figure 2.14: LFA coverage in each iteration of different heuristics in the link-protecting case for AT&T with SRG density $\delta = 0.5$, and node-protecting case for Germany topology with $\delta = 0.9$

be that the new links we add never appear in an SRG, and therefore can provide massive improvement in LFA coverage. Similar observations can be made for the node-protecting case as well. We also observe that the rank of the algorithms, in terms of efficiency, does not change under the SRG model: the MSBT algorithm is still the most efficient.

The results for the LFA graph improvement problem under the SRG model for some selected topologies are shown in Fig. 2.14. The observations are similar as in the non-SRG case: when the goal is merely to improve the coverage instead of shooting for 100% the LJC algorithm is clearly the best algorithmic strategy.

As a summary, we can conclude that our results verify the potential of heuristic algorithms for solving the minLFA$_{\mathrm{SRG}}$ problem. The proposed algorithms, combined with the construction we have shown, are able to efficiently approximate the optimal solution even though the problem is NP-complete.

2.4 Application of Results

Below, we outline how our results contribute to other related studies. First of all, the work around network optimization to increase the LFA coverage was conducted in partnership with Ericsson Research, Hungary, our industrial partner. Our results were utilized during the development of an internal application that is capable to analyze and optimize LFA characteristics in operator networks.

Besides, our findings are integrated into a few scientific reports. The IEEE Communications Surveys and Tutorials aims to cover all aspects of telecommunication technology. In a study submitted to this forum, Chiesa *et al.* [57] presents a systematic tutorial-like overview of packet-based fast-recovery mechanisms in the data plane. Our results on LFA graph extension contribute to the section of fast re-route proposals where different network optimization methods are discussed.

The Future Internet Assembly (FIA) [58] is a special issue of the Computer Sciences book series that aims to report the latest results of information technology research and development. The authors of [59] give a comprehensive overview on the Loop-Free Alternates and the related network optimization methods, such as the LFA graph extension problem. Incorporating our results, the study presents a combined model of *graph extension* and *LFA cost optimization*, where complement edges are added gradually followed by a step of link cost optimization in the network. The results suggest that the combined algorithm can significantly reduce the number of additional links (on average by more than 50%) for achieving full LFA protection.

2.5 Related Work

Finally, we continue by discussing how our results parallel or differ from prior survey research.

The idea behind *Remote LFA* [60] is to extend the set of eligible LFAs from adjacent nodes to next-hops that are more than one hop away. The solution relies on tunnels that connect unprotected source nodes with rLFA instances that can lead packets to the destination along loop-free shortest paths. To achieve this goal, one need to define a new control plane protocol or use existing ones (e.g. MPLS/LDP) that unfortunately brings back the management complexity.

It is noteworthy that the authors of [53] recognized that LFA already holds potential in its simplest form. The idea is to shift the focus from device or protocol development to the field of network optimization where the goal is to shape the layout of the topology so that LFA coverage improves. Accordingly, the main goal of the study was to identify the relationship between the layout of the topology and its LFA coverage, and they found that in uniform cost networks *rings with even number of nodes have the worst LFA coverage.* It was also found that extending the network with complement edges leads to increase in LFA coverage and the complexity of this task is substantial.

Last but not least, [61] aims to optimize IGP link costs to raise the number of LFA protected node pairs. The paper argues that by sacrificing forwarding efficiency there could be a significant gain (more than 50%) in the level of protection, although the problem is NP-complete. Unfortunately, this method only seems to provide acceptable results in denser networks otherwise most of the original shortest path had to be redesigned.

Chapter 3

Node Virtualization for IP Level Resilience

So far now we have seen that to achieve full failure coverage with LFA, one has to completely re-engineer the network around LFA-compliant design patterns. But is there any alternative way that solves the same problem without touching the physical topology? In this chapter, we show how to construct a virtual overlay on top of the physical topology that brings LFA coverage to perfect.

3.1 Background

Router virtualization is a technique used for sharing the resource of a single IP routing device between multiple virtual instances [62]. Accordingly, virtual routers are indistinguishable from physical routers, each instance having its own forwarding and control planes, which allows us to assign virtual routers as LFAs to routers that originally did not have one. Network virtualization was primarily introduced to overcome Internet ossification and increase diversity, to provide isolated environments for experimental protocols, and to improve utilization and security in provider networks, all in all, to let operators to extract larger profit from their valuable infrastructure [63, 64].

Our idea is to provide "virtual LFAs" for the physical nodes that would go unprotected otherwise. Adding virtual nodes and links to an underlying link-state routing protocol is a well recognized technique to implement new functionalities, such as to enable better load balancing, traffic engineering, and backup routes [65]. These solutions have an appealing *plug-n-play* quality, in that they can be deployed without modification to expensive router ASICs as a simple one-time software upgrade.

3.2 Model and Notation

As a consequence of introducing virtual nodes to the topology, we need to revise the model we gave in Sec. 2.2.1. The main goal is to minimize the interference with the normal operation of the network, and only involve the virtual layer in packet forwarding when absolutely necessary. This guarantees that packets do not take excess detours, helps break down management complexity, and eases debugging data plane misconfiguration. Similar to Sec. 2.2.1, we model the network with a weighted, symmetric directed graph but this time we denote the physical topology, or *substrate*, with $G_S = (V_S, E_S)$. We make the following **assumptions on physical and virtual topology**.

In our model G_S is a subgraph of G_V. Nodes in $G_V \setminus G_S$ are called *virtual nodes* and links are called *virtual links*. Mark the default node for a physical router $v \in V_S$ and denote its virtual nodes by $v^i \in V_V \setminus V_S$ for $i = 1, \dots, k_v$ where k_v denotes the number of virtual instances running on the particular physical router. Let $k_{\max}$ denote the maximal number of virtual instances a router may have. In addition, denote the set of neighbors of some node $v \in V_S$ in G_S by $N_S(v)$. Similarly, $N_V(v)$ denotes the neighbors of some $v \in V_V$ in G_V.

Traffic flows in the default layer along the default shortest paths. Traffic only enters a virtual router when a failure shows up, and so virtual routers serve exclusively as LFAs for nodes not protected in the physical topology. To achieve this, the cost of virtual links is set so that they never appear in any $u \to v$ shortest path in G_V for any $(u, v) \in V_S \times V_S$, $u \neq v$.

Virtual links connect physically connected nodes. Virtual links are provisioned between nodes that are adjacent in the substrate, or inside the same physical router:

$$\forall (v^i, u^j) \in E_V \setminus E_S : \{(v, u) \in E_S \text{ or } v = u)\} \ , \tag{3.1}$$

where $i = 1, \dots, k_v$ and $j = 1, \dots, k_u$. The reasons for this assumption are manifold.

First, as virtual links never span multi-hop paths, they are easy to provision as layer-2 virtual links (say, Ethernet VLANs). Such connections often do not even require distinct IP addresses. This minimizes impact on the IP layer and eliminates much of the configuration overhead and MTU issues that plague tunnel-based IPFRR mechanisms [27]. Additionally, layer-2 connections are free from the limitations of layer-3 tunnels, which are bound to shortest paths. Finally, two virtual links now belong to the same SRG if and only if they share the same physical link, which would not hold over multi-hop tunnels.

Single link failures in the physical network. As we found previously, single link failures have been shown to constitute the major portion of unplanned outages, therefore we stick to this scenario and we also ignore node failures; these can be incorporated into the model with little extra effort. Note that even a single physical link failure manifests itself as multiple simultaneous failures in the virtual topology for which we apply the already familiar *local SRG* definition. By introducing virtual links on top of physical ones we associate with each link $(u, v) \in E_S$ local SRGs at both end nodes composed of all the links provisioned on the same physical link as e, formally

- SRG at v is $S_{(v,u)} = \{(v, u), (v, u^1), \dots, (v, u^{k_u})\}$,
- SRG at u is $S_{(u,v)} = \{(u, v), (u, v^1), \dots, (u, v^{k_v})\}$.

3.3 New Results

We show in this section how to *provision a virtual overlay* on top of the physical topology that makes LFA coverage of any given network perfect.

3.3.1 Motivation

A noteworthy result of Sec. 2.3.6 was the relatively high initial LFA coverage of the input graphs. The promise to close this small gap to perfect protection, is therefore seems to be within reach. In addition, achieving this **without touching the physical topology** and simply execute a software upgrade makes our approach rather appealing. Fortunately, virtualization allows to instantiate appliances on demand without the need for costly purchase and integration of additional hardware that

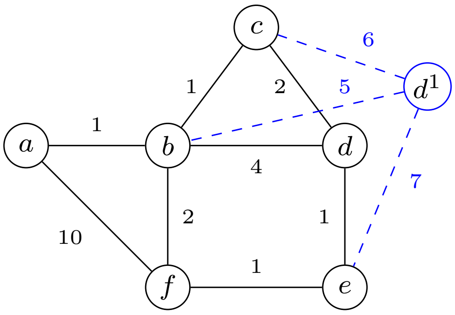


Figure 3.1: Extending the sample network with a virtual router for improving LFA coverage. Virtual elements are colored with blue.

makes a dramatic change nowadays in the landscape of telecommunications industry [66].

The idea is demonstrated on a sample topology in Fig. 3.1. Traffic between node c and a follows the single shortest path, $c \to b \to a$. Assuming the failure of link (c, b), packets get dropped until the IGP restoration process is completed since there is no link-protecting $c \to a$ LFA in the network. This is because d, the only possible LFA candidate, is an upstream of c to a. Now, if we provision a *virtual router*, d^1, on top of the hardware of d and duplicate some d's physical links, then we can assign costs to these links in a way as to ensure that d^1 can eventually become an LFA from c to a. In our example if the link (c, b) goes down, then c can still use the $d^1 \to b \to a$ path to reach its destination, a. Observe that not just that d^1 is now an LFA from c to a, but it also protects node pair $e \to f$.

Therefore, our first goal is to understand **if virtual overlays are capable of providing perfect LFA protection for any arbitrary input graph**. Next, **we define how to construct such an overlay** and we also characterize the complexity of this task. Furthermore, our goal is *to identify and compare different cost assignment strategies on the virtual links* to be able to maximize the number of protected node pairs with minimum number of virtual devices. Then, by having the virtual layer in place, *we evaluate the resultant ratio of virtual vs. physical nodes* and *the average path length a packet will take in the virtual layer upon a failure.*

3.3.2 Problem Formulation

First of all, we augment the LFA definition to the case of virtual layers.

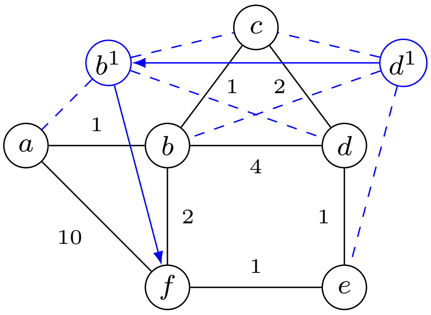


Figure 3.2: A sample network with a possible LFA loop (the shortest path is marked with arrows in the virtual layer): suppose that b is about to send packets to f. If, for some reason, link (b, f) goes down, b may choose to redirect its traffic to the LFA d^1. However, said traffic will never arrive to f as the $d^1 \to f$ detour degrades into the LFA loop $b - d^1 - b^1 - c - b$. Here, b^1 also switches to its LFA c realizing that its link to f has disappeared due the physical failure which node d^1 is unaware of.

Definition 8. *For source s, destination d, and $s \to d$ next-hop e, node n is an SRG-disjoint link-protecting $s \to d$* LFA *if*

i) $n \in N_V(s)$ *and* $n \neq e$, *and*

ii) $\text{dist}(n, d) < \text{dist}(n, s) + \text{dist}(s, d)$, *and*

iii) $(s, n) \notin S_{s,e}$ *(local SRG condition), and*

iv) $\text{dist}(n, d) < \text{dist}(n, s^i) + \text{dist}(s^i, d)$ *for* $i = 1, \ldots, k_s$.

Here, *iv)* requires that n is an LFA with respect to *all* virtual neighbors. Since current LFA implementations are restricted to *i)*, *ii)*, and *iii)*, our virtual overlay construction algorithms need to be designed so that *iv)* automatically fulfill. Without asking for *iv)*, the risk of creating forwarding loops is significant as bouncing packets between the virtual and physical layers may be blindly directed back to the origin. We show an example in Fig. 3.2 for this problem where, after a failure, the traffic ends up in an LFA loop. Therefore, to completely rule out this phenomena we require *iv)* by prohibiting “cascade LFAs” in the virtual layer.

Proposition 2. *In our model, a packet can be deflected to an LFA only at most once during its journey from the source to the destination.*

According to our experience, this limitation does not have any perceptible degradation in the resultant LFA coverage. With these notations in place, we can now pose the *Resilient IP Overlay Design* (RIOD) problem. Here, the task is to compute the overlay that maximizes LFA-coverage, using only a given number of virtual routers. In addition, we also allow to limit the set of routers that can host virtual instances.

Definition 9. $\mathrm{RIOD}(G_S, c, U, k, \eta_{\min})$*: given a graph* $G_S = (V_S, E_S)$*, link costs* c*, node set* $U \subseteq V_S$*, and positive integer* k*, design a graph* $G_V = (V_V, E_V)$ *and link costs* c_V *so that:*

- $V_S \subseteq V_V$ *and virtual nodes provisioned only inside* U*,*
- $E_S \subseteq E_V$ *and virtual links are only between physically connected routers* (see Eq. (3.1))*,*
- *shortest paths between node pairs in* V_S *do not change* (the substrate is unaltered)*,*
- $|V_V \setminus V_S| \leq k$ (no more than k virtual instances)*, and*
- $\eta(G_V, c_V) \geq \eta_{\min}$ (the LFA coverage is a least $\eta_{\min}$)*.*

One ultimate goal would be to build an entire virtual topology in one step so that each node-pair in the substrate becomes LFA-protected. In [67], we show a construction that achieves this goal in certain circumstances. A somewhat less ambitious task is $\mathrm{RIOD}(G_S, c, \{v\}, 1, \eta_{\min})$ where we add a *single virtual router* to a selected node v in order to merely *maximize LFA coverage* along the way instead of aiming for full coverage. We shall refer to this special case of RIOD as the *LFA Virtual Router Augmentation Problem* $\mathrm{LFAVirt}(G_S, c, v)$ [68].

3.3.3 The Solvability and Complexity of RIOD

Next, we address the problem of building an LFA-optimized overlay under the model assumptions we have. The model basically asks for augmenting a physical topology with virtual routers and set the cost on the resultant virtual links in a way as to maximize LFA coverage. Let us first show that full LFA coverage can always be achieved by RIOD.

Theorem 4. *For a given* 2*-connected graph* G_S *with positive costs* c *there always exists an overlay* G_V *and cost setting* c_V *that solves* $\mathrm{RIOD}(G_S, c, V_S, \infty)$ *with* $\eta(G_V, c_V) = 1$*.*

Proof. We show that, given any unprotected node-pair $s \to d$, there is a proper set of virtual nodes whose addition will create a link-protecting SRLG-disjoint $s \to d$ LFA.

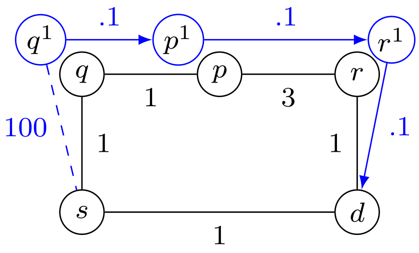


Figure 3.3: A sample network with a“virtual tunnel” between s and d.

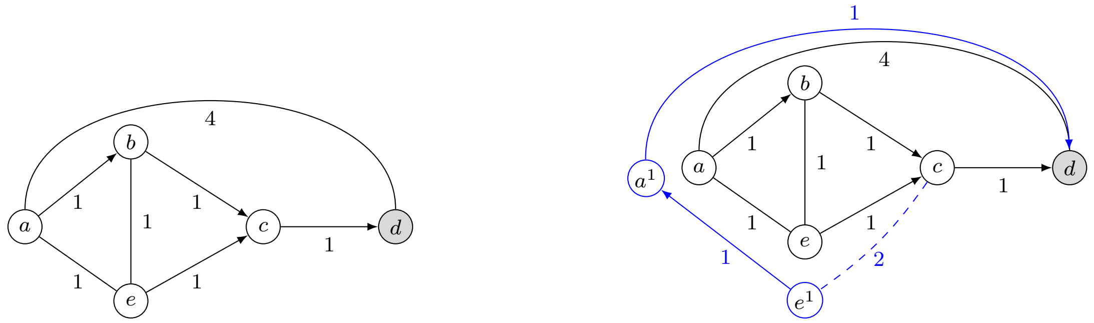
It follows that if we apply this step to each unprotected node-pair, then full LFA-coverage eventually reaches. We protect the $s \to d$ pair by provisioning a *“virtual tunnel”* between s and d that provides a detour for s bypassing its failed next-hop (Fig. 3.3). Let $s - q - \ldots - r - d$ be an $s \to d$ path disjoint from the $s \to d$ next-hop (such a path is guaranteed to exist as G_S is 2-connected). Create a virtual node for each node between q and r and denote the new virtual router on q by q^1 and the one on r by r^1. Connect s to q^1 and r^1 to d and set the link cost on (s, q^1) “high” (say, larger than the length of the longest shortest path) and at the rest of the virtual links to the lowest possible. As one easily checks, q^1 is now an $s - d$ LFA. We still need to show that *(iv))* holds, but this is guaranteed as there are only two entry points to the virtual tunnel, q^1 and r^1, and q^1 is protected by the local SRLG at s (as of *(iii))*) and r^1 is never an LFA due to its low cost. □

The first immediate question is whether the RIOD problem is tractable. The following statement gives a negative answer.

Theorem 5. *$RIOD(G_S, c, U, k, \eta_{min})$ is NP-complete.*

Proof. In [68] the authors showed that adding a single virtual node to the network that maximizes the LFA coverage (LFAVirt(G_S, c, v)) is already NP-complete. Since RIOD tries to achieve the same goal by utilizing an *extended set* of virtual nodes, we can conclude that the complexity of the simplest subcomponent applies to the complexity of RIOD as well. □

Unfortunately, today’s large size of IP backbones can easily trigger unsolvable ILP instances. Correspondingly, we go on to design a heuristic algorithm that, depending on a setting of a simple configuration parameter, is *either optimal or it is guaranteed*



(a) LFAVirt stuck with a single node to provide a $c \to d$ LFA

(b) The e^1-a^1 tunnel protects $c \to d$

Figure 3.4: A graph on the left, where LFA coverage cannot be increased by adding a single virtual node. The d-rooted shortest path is marked with arrows. On the right it is shown that by using 2 virtual nodes, d gets protected.

to terminate in polynomial time, and it is possible to efficiently balance between the two according to the preferences of the operator.

3.3.4 Heuristic Algorithms to the RIOD Problem

The main idea in our heuristics is to iteratively add new virtual nodes to the network until full LFA coverage is achieved. First, let us focus on the special case LFAVirt(G_S, c, v) where a single virtual node v^1 is added to v and the task is to set the link costs so that LFA coverage increases the most. Clearly, this greedy approach would curtail the computational complexity significantly, but on the negative side this simple logic may stuck in a local maximum in certain cases.

Theorem 6. *There exist cases where LFA coverage cannot be increased by adding only a single virtual node.*

Proof. We show an example in Fig. 3.4a. One can verify that the only unprotected node-pair is $c \to d$, so the LFA coverage of the network is $\eta = \frac{19}{20}$. Additionally, all the 2-hop neighbors of c are upstream for d, and because LFAVirt(G_S, c, v) provisions only a single virtual router in each step the furthest it can reach with a virtual tunnel is a, which is also upstream. Hence, any greedy algorithm that adds only a single node in each step terminates with $\eta < 1$. □

It may be tempting to believe that our example is a pathologic case due to the large cost of the (a, d) link. This, however, is not the case, as one can easily show unit-cost counter-examples similar to the one in Fig. 3.4a (e.g., by substituting the (a, d) link with a long chain of unit-cost links).

Therefore, we introduce an extended version of LFAVirt, the *General LFA Virtual Router Augmentation Problem* (GLFAVirt), whereby we may instantiate multiple virtual nodes in each step if needed. We continue by giving the formal definition of GLFAVirt, and we integrate it into a greedy heuristic framework that solves the fully fledged $RIOD(G_S, c, U, \infty, \eta_{\min})$ problem.

Basic Heuristic

As it turns out, augmenting the graph with only a single virtual node in each iteration might be too restrictive. Instead, one might try to instantiate two virtual routers when adding only a single one did not help, then try three virtual routers at once, etc. This observation is reflected in the below definition of *connected l-sets*.

Definition 10. *For a graph G_S, call a set of nodes in $U_l \in V_S$ a* connected l-set *if the induced subgraph of G_S spanned by U_l is connected and $|U_l| = l$. We denote the set of virtual nodes on top of U_l with U_l'.*

The generalized LFAVirt problem, GLFAVirt, is then able to take a connected l-set as an input parameter, and returns the cost setting of the virtual links that maximizes the LFA-coverage. Formally:

Definition 11. GLFAVirt(G_S, c_S, U_l, j): *Given a substrate $G_S(V_S, E_S)$ with link costs c_S, a positive integer j, and a connected l-set $U_l \subseteq V_S$ for any $l \geq 1$ integer, inducing the virtual topology $G_V(V_V, E_V)$ with a virtual nodes $V_V = V_S \cup U_l'$, $E_V \subseteq E_S \cup \{(v, u) : v \in U_l', u \in N_S(v)\}$, is there a cost setting c_V on E_V so that (i) the link costs and shortest paths in G_S do not change and (ii) # protected (s, d) pairs $\geq j$, $s, d \in V_S \times V_S$?*

Our heuristic is then based on simply trying increasingly larger connected sets of virtual nodes until LFA coverage eventually improves. Note that, by Theorem 4, there is a sufficiently large connected l-set for which at least one node-pair will gain a new LFA, and therefore this modification to the algorithm is guaranteed to terminate in finite steps with an improved LFA coverage. In every iteration we greedily select the set of nodes whose addition to the network increases LFA coverage the most. When the coverage stops increasing we terminate the execution. Algorithm 6 implements these ideas.

Algorithm 6 Greedy alg. for RIOD($G_S, c, U, \infty, \eta_{min}$)

```
while $\eta_{min} > \eta(G_S, c)$ do
    for each $l = 1, \dots, k$ do
        for each connected $l$-set $U_l \subseteq U$ do
            $(c_{U_l}, \eta_{U_l}) \leftarrow$ solve GLFAVirt$(G_S, c, U_l)$
        end for
        $(U'_l, \eta') \leftarrow$ choose $U_l \in U$ that maximizes $\eta_{U_l}$
        if $\eta' > \eta$ then add $U'_l$ to $G_V$ and set costs to $c_{U'_l}$
            break
        end if
    end for
end while
```

Note that the algorithm is parametrized on an integer k, which allows to set an upper bound on the maximum size of the connected l-sets examined, and hence on the running time.

There still remains the problem of how to solve the general form of the LFA Virtual Router Augmentation problem GLFAVirt(G_S, c, U_l). Here, we need to assign an entire "island of virtual nodes" on a connected set U_l. Let neigh(U_l) denote the neighbors of nodes in U_l that are outside U_l. Suppose that we are about to solve GLFAVirt(G_S, c_S, U_l) by provisioning a set of virtual routers U'_l on U_l, with each $u \in U_l$ hosting a single virtual instance u'.

The main idea of our heuristics is to set a single exit point for the virtual nodes in U'_l, that is, to let all traffic that enters U'_l through LFAs leave via a *single exit link* (v', g). Thus, we create a virtual router at each node of U_l and we connect these to each other with small cost, plus we connect these nodes to all nodes in neigh(U_l) with a large cost except for the virtual link to the exit link (v', g) which is again set to low cost, just like in Fig. 3.3. It is now trivial to check that this setting indeed yields that U'_l has a single exit node: g.

This is a simple yet efficient method, with the main drawback that for each $u \in$ neigh(U_l) we need to evaluate the LFA coverage. The LFA coverage is computed according to Eq. (2.1) and verified by a simple LFA-aware packet tracer (Prop. 2) which needs evaluating every edge-node pair: $(s, n) \in E_V$ and $d \in V_D$. Recall,

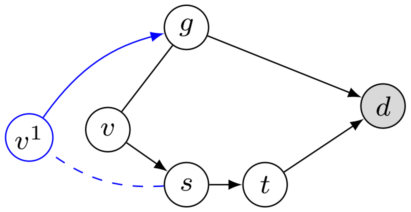


Figure 3.5: Illustration for *escape nodes*. Suppose that s is about to send packets to d, and $U_1 = \{v\}$. We call g an escape node, if it can be used as an exit point of the virtual layer without the risk that it would direct the traffic to a blackhole.

$s \in V_S$, thus the number of possible (s, n) LFA candidate links are at most $k_{max}n$ where $n = |V_S|$. This is performed for each node $u \in \text{neigh}(U_l)$; thus, the process takes $O(k_{max}n^2 \cdot |\,\text{neigh}(U_l)|)$ steps in total each time virtual nodes U_l are added.

Reducing the running time

The idea of allowing only a single exit point from U'_l reduces the complexity of GLFAVirt(G_S, c_S, U_l) significantly. However, there are other computational heavy steps in Alg. 6, such as:

- evaluation of LFA coverage in each step
- selection of all possible connected l-sets

First, whenever a virtual node is added to the network, LFA coverage calculation for all node pairs is launched. **To speed it up we switch to incremental evaluation**. To do so, we keep track the set of *eligible node-pairs* $\mathcal{L}$ that can gain an LFA. Clearly, a virtual router u' can provide LFA only if it is a neighbour of the source node. Let $\mathcal{L}_{U_l} \subseteq \mathcal{L}$ denote the set of eligible node-pairs with source node adjacent with U_l, formally $\mathcal{L}_{U_l} \subseteq \mathcal{L}|(s, d) \in \mathcal{L}, s \in \text{neigh}(U_l)$. In other words, the new virtual nodes U_l can provide LFA to node pairs $\mathcal{L}_{U_l}$; thus $\frac{|\mathcal{L}_{U_l}|}{n(n-1)}$ is the upper bound in the increase of $\eta(G)$ after adding U_l. This measure helps in selecting a proper virtual nodes U_l to add.

Let $(s, d) \in \mathcal{L}_{U_l}$ be the set of source-destination pairs that can gain an LFA when new virtual nodes U_l are added to the physical node v. We seek nodes $g \in \text{neigh}(U_l)$ that can provide LFA to $s-d$ by U_l. Therefore, we pre-calculate the following set: for each $(s, d) \in \mathcal{L}_{U_l}$ a set of *escape nodes* $\mathcal{E}_{s \to d}$ consisting of the nodes which, if chosen as the only exit link of U_l to d, would render a virtual node $v' \in U_l$ an $s \to d$ LFA.

Computing $\mathcal{E}_{s\to d}$ for all $(s,d) \in \mathcal{L}_{U_l}$ takes $O(|\operatorname{neigh}(U_l)| \cdot |\mathcal{L}_{U_l}| \cdot k_{max})$ steps in total. Based on this, we can select optimal g as follows

$$g = \operatorname*{argmax}_{g' \in \operatorname{neigh}(U_l)} |(s,d) \in \mathcal{L}_{U_l} : g' \in \mathcal{E}_{sd}| \quad . \tag{3.2}$$

Second, we reduce the complexity of selecting connected l-sets that is the other bottleneck of Alg. 6. To obtain all possible connected l-sets in an arbitrary network takes $O(n^k)$ steps, where $l = 1 \dots k$. When running time is of no concern then k can be set to n, in which case we are guaranteed to obtain a fully-protected overlay. Unfortunately, this overhead might not always be acceptable. Thus, **by calculating only a reasonable subset of connected l-sets**, say **shortest path slices of rank** l, we can reduce $O(n^k)$ to $O(n^2)$.

Definition 12. *For a graph G_S, call a set of nodes in $U_l \subseteq V_S$ a* shortest path slice of rank l *if the induced subgraph of G_S spanned by U_l is created by shortest paths of G_S and $|U_l| = l$.*

By choosing the maximum length of the shortest path slices small, it results strictly polynomial running time. As we will see later, fixing $k = 3$ results in perfect LFA coverage in most of the practical cases. Incorporating the enlisted optimization steps, the following statement is now obvious:

Theorem 7. *There exists a set of heuristic algorithms that terminates Alg. 6 in polynomial running time, at most $O(n^5)$ steps.*

Proof. The computational complexity of Alg. 6 is $O(n^3 + n^{k+3})$. Here, $O(n^3)$ comes from the all-pairs shortest path problem needed be solved to obtain dist(.) and $O(n^k)$ is the number of connected l-sets U_l of size at most k where $k = 3$ due to Def. 12. The basic heuristic solves GLFAVirt(G_S, c_S, U_l) for each U_l also in $O(n^3)$, as $\mathcal{L}$ and $\mathcal{L}_{U_l}$ contain $O(n^2)$ elements and $\mathcal{E}_{sd}$ contain $O(n)$ elements, and each can be calculated in $O(n^2)$ steps. □

Accordingly, the theoretical worst-case complexity of Alg. 6 is $O(n^5)$ steps, however, in practice we found the all-pairs-shortest path problem to dominate running time and hence $O(n^2)$ to be a more reasonable complexity characterization.

Integer Linear Program for GLFAVirt(G_S, c_S, U_l)

Since the GLFAVirt(G_S, c_S, U_l) problem is NP-hard, we provide an Integer Linear Program (ILP) that will serve as the baseline when evaluating the heuristic. The formulation is based on the definition of escape nodes discussed in the previous subsection; however, instead of selecting one escape node, the ILP computes the optimal virtual link costs so that the most escape nodes become next-hops for the new virtual node $v' \in U_l$ and hence LFA-coverage is maximized when adding U_l.

The ILP is built around the following two constraints:

1. We want to assign at least one escape node as the next-hop of v' towards d, as then v' will provide a new LFA to $s \to d$. For this, we need to set link costs c_V such that

$$\operatorname{dist}(v', d) = c_V(v', g) + \operatorname{dist}(g, d) \text{ for some } g \in \mathcal{E}_{sd} \ . \tag{3.3}$$

2. Besides we need to ensure the Def. 8.*(iv)* condition holds

$$\operatorname{dist}(v', d) < c_V(v', s^i) + \operatorname{dist}(s^i, d) \text{ for all } i = 1, \dots, k_s. \tag{3.4}$$

The ILP is based on the idea that eligible node pairs and the respective escape nodes can be pre-computed statically, so $\mathcal{L}$, $\mathcal{L}_v$, and $\mathcal{E}_{sd}$ can be generated offline. Hence, in the course of the optimization we only need to take care of satisfying (3.3) and (3.4).

The variables of the ILP are as follows:

- The binary variable $x_n : n \in N_V(v)$ tells whether to provision the virtual link (v^k, n): $x_n = 1$ if (v^k, n) is a new virtual link, and zero otherwise.
- The binary variable $y_{s,d} : s, d \in \mathcal{L}$ marks whether $s \to d$ has obtained an LFA: $y_{s,d} = 1$ if $s \to d$ has LFA after adding v^k, and zero otherwise.
- The binary variable $z_{g,s,d} : s, d \in \mathcal{L}, g \in \mathcal{E}_{sd}$ is set so that $z_{g,s,d} = 1$ if g is the next-hop of v^k, zero otherwise.
- The non-negative real variable $c_n : n \in N_S(v)$ represents the cost $c_V(v^k, n)$ of the virtual link (v^k, n). We require that $c_n \geq c_S(v, n) + C$ where C is a problem parameter, to ensure that paths via v^k are longer than the default shortest paths. In the rest of this paper, we set $C = 1$.

- Finally, the non-negative real variable $\delta_u : u \in V_S \setminus \{v, v^k\}$ denotes the shortest path distance from v^k to u.

Consider the ILP below (the role of parameters K and ϵ will be made clear soon).

$$\begin{array}{lllr}
\text{maximize} & \sum\limits_{s,d \in \mathcal{L}} y_{s,d} - \epsilon \sum\limits_{n \in N_S(v)} (c_n + x_n) & & \\
\text{subj. to} & y_{s,d} \leq x_s,\ \ z_{g,s,d} \leq\ x_g & s, d \in \mathcal{L}_v, g \in \mathcal{E}_{sd} & (1) \\
& y_{s,d} \leq \sum\limits_{g \in \mathcal{E}_{sd}} z_{g,s,d} & s, d \in \mathcal{L} & (2) \\
& \delta_d \leq \mathrm{dist}(g, d) + \mathrm{dist}(u_s, u_g) + c_g & s, d \in \mathcal{L}_v, g \in \mathcal{E}_{sd}, & \\
& \qquad\qquad +K(1 - z_{g,s,d}) & u_s, u_g \in U_l & (3) \\
& \delta_d + C \leq c_{s^i} + \mathrm{dist}(s^i, d) + \mathrm{dist}(u_s, u_g) & & \\
& \qquad\qquad +K(1 - z_{g,s,d}) & i = 1, \dots, s_k & (4) \\
& c_n \geq c_S(v, n) + C & n \in N_S(v) & (5) \\
& x_n, y_{s,d}, z_{g,s,d} \in \{0, 1\},\ c_n \geq 0 & & (6)
\end{array}$$

The objective function maximizes the number of LFAs the new virtual node v' gives rise to. Parameter ϵ is a small constant, which ensures that the optimization favors the solution with the smallest link costs and the fewest virtual links. The first constraint in (1) states that v' can only become an LFA for s if the virtual link (s, v') is present. Similarly, $z_{g,s,d} \leq x_g$ expresses that we can only set g as next-hop for v' if the virtual link (v', g) is provisioned.

Constraints (2) and (3) correspond to the escape node condition (3.3) for each $s \to d$ pair in $\mathcal{L}_v$. In particular, (3) will set the shortest path distance from v' to d according to whether the escape node $g \in \mathcal{E}_{sd}$ is chosen as the next-hop for v' to d. For this, we have to ensure that the shortest path in the virtual layer does not override the one on the physical path. Therefore we set a lower bound on the costs of virtual links within the tunnel with $dist(u_s, u_g)$, where $u_s, u_g \in U_l$ and $u_s \in N_V(s)$, and similarly $u_g \in N_V(g)$ representing the two endpoints of the tunnel. If $z_{g,s,d} = 0$, i.e., if g is not the next-hop then the constraint is inactive, while if $z_{g,s,d} = 1$ then the constraint is active and sets δ_d and c_g according to (3.3). To switch between the active and inactive states, we use the large constant $K \gg C + \max_{(s,d) \in V_S \times V_S} \text{dist}(s, d)$. Furthermore, (2) sets an $s \to d$ pair protected, if at least one escape node has been selected as the next-hop for v' towards d.

Constraint (4) stands for the Def. 8.*(iv)* condition (3.4) for eligible $s \to d$ pairs. The constraint is only active when both (s, v') and (v', g) virtual links are present, i.e., $x_s = 1$ and $x_g = 1$. In this case, it sets c_g to prevent s^i to become a next-hop for v' to d according to (3.4) for $i = 1, \ldots, s_k$. Finally, the domain of the variables is set in (5)–(6).

After solving the ILP, the virtual topology is constructed by augmenting the substrate with the virtual node set U_l' and the virtual links $(u', n) : x_n = 1$ with cost c_n for all $n \in \text{neigh}(U_l)$.

3.3.5 Numerical Evaluations

We evaluate the performance of the heuristic algorithms in the followings. Our major interest is to check whether the heuristic performs close to the ILP, measured by the LFA coverage metric, η, as an increasing number of virtual routers is added to the network. Obviously, we do not expect it to outperform the optimal solution, but we hope that the performance is not prohibitively worse and the improved running time makes up for the penalty.

The parameter l was set to 3 (i.e. U_3) meaning that the algorithm tries to provision node sets with size up to 3 in each step. As we observed in our initial measurements, there is marginal difference in the outcome of using all possible connected l-sets or shortest path slices, therefore we settle for the computationally less demanding shortest path slice based selection. The detailed results are summarized in Table 3.1.

Here, the values refer to the LFA coverage (η) attained by the heuristic as an increasing number of virtual nodes are provisioned in the network. The most important observations are as follows. First, the initial LFA coverage was 70-90% in the examined topologies and it can be easily boosted up to 95%, just by introducing a single virtual instance for half of the physical nodes. Second, the heuristic barely overshoots the ILP in the cases when the number of virtual nodes is less than, or equal to $|V_S|$. The last two columns show that if we do not maximize the amount of virtual nodes the algorithm terminates with $\eta = 1$ in most of the cases, and there are

Table 3.1: Results of Alg. 6 for RIOD(G_S, c_S, U_3, k) using shortest path slices: topology name, initial LFA coverage (η_0; [%]), attained LFA coverage (η; [%]) and running time (t; [sec]) when provisioning $25-50$ and 100% of the physical nodes respectively, the final LFA coverage (η_∞) with the required execution time (t_∞) and the ratio of virtual nodes in the final state (v_∞; [%]). Slashed values indicate the results of the ILP in case there is a difference compared to the heuristic.

		$k < 0.25\|V_S\|$		$k < 0.5\|V_S\|$		$k < \|V_S\|$		$k = \infty$		
Topology	η_0	η	t	η	t	η	t	η_∞	t_∞	v_∞
Abilene	0.61	0.70	0.1	0.80/0.81	0.2	0.92/0.96	0.5	1	1.5	1.72/1.36
Germany	0.69	0.83/0.84	0.7	0.90/0.91	1.6	0.96/0.98	4.8	1	12.1	1.58/1.29
BtEurope	0.96	0.98	3.4	0.99	8.5	1	13.2	1	13.2	0.70/0.58
AS6461	0.93	0.99	2.6	1	5.4	1	5.4	1	5.4	0.35/0.47*
Internet_MCI	0.95	0.98	1.8	0.99	6.3	1	7.7	1	7.7	0.55
AS1755	0.87	0.96	2.1	0.98/0.99	6.8	1	15.6	1	15.6	0.88/0.61
ChinaTelecom	0.95	0.98/0.99	19.5	0.99/1	49.9	1	60.7	1	60.7	0.65/0.45
AS3967	0.78	0.95	3.3	0.99	9.2	1	14.0	1	14.0	0.61
BellSouth	0.79	0.96/0.97	12.4	0.98/0.99	44.9	1	149	1	149	0.85/0.52
ATnT	0.82	0.92	6.2	0.97/0.98	19.1	1	63.3	1	63.3	0.95/0.81
NSF	0.86	0.93	6.1	0.97/0.98	18.1	1	46.9	1	46.9	0.88/0.80
BICS	0.76	0.91/0.92	3.8	0.96/0.97	11.6	0.99	42.1	0.99	52.1	1.25/1.03
AS3257	0.92	0.99	50.7	1	160	1	160	1	160	0.40/0.44*
AS1239	0.87	0.98	54.0	0.99/1	109	1	130	1	130	0.60/0.46
Arnes	0.83	0.97	10.1	0.98/0.99	33.5	0.99/1	134	1	143	1.09/0.96
Geant	0.83	0.96	8.7	0.98/0.99	28.2	0.99/1	79.6	0.99/1	86.1	1.09/1
Italy	0.78	0.89	24.3	0.94/0.96	58.0	0.99	178	1	367	1.57/1.33
BtN.America	0.83	0.96	65.6	0.99	211	1	407	1	407	0.80/0.63
BellCanada	0.61	0.80	10.2	0.90/0.91	25.9	0.98/0.99	73.9	1	169	1.82/1.48
Germany_50	0.90	0.96/0.97	60.1	0.99	168	0.99/1	412	1	430	1.04/0.78
Deltacom	0.63	0.85/n.a	476	0.96/n.a	950	0.99/n.a	2494	0.99/n.a	8323	2.04/n.a
Average	0.81	0.92/0.94	39.1	0.96/0.97	92.2	0.99/0.995	214.0	0.99/0.99	507.6	1.02/0.83

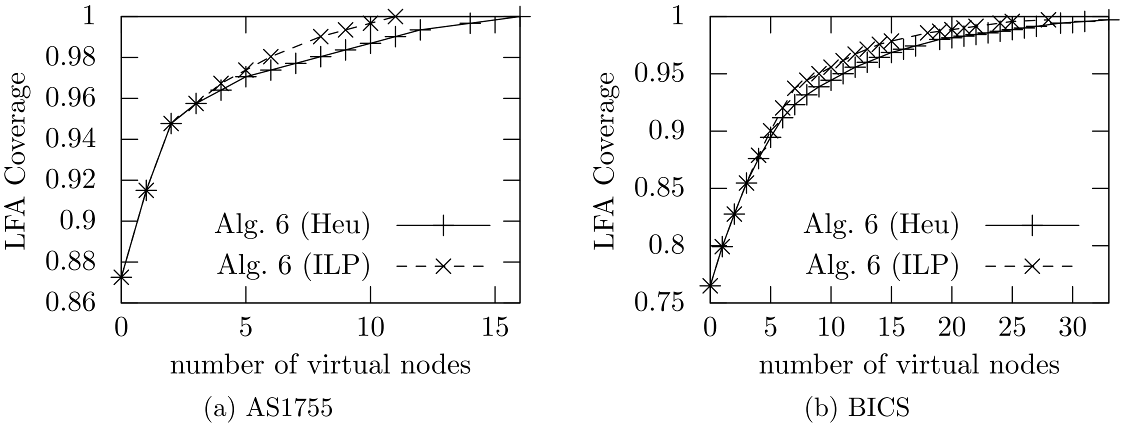
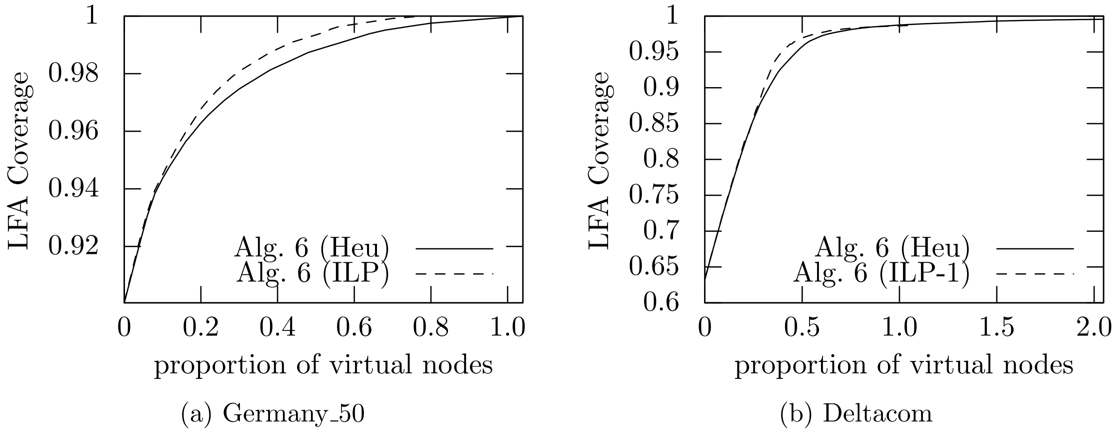


Figure 3.6: Progression of LFA coverage in small and middle-sized networks.

only 3(!) networks whereby it gets stuck with the setting $l = 3$.

The price that we pay for reaching this final state is given below as v_∞. As an average the heuristic shows some 19% overhead compared to the the ILP when adding $|V_S|$ virtual nodes, however it executes 10 times faster. The reason is that the LFA coverage improvement has a logarithmic trend, so if the goal is to solely improve LFA protection to a certain level then a couple of new nodes are usually enough. In contrast, to achieve full protection, we need to provide alternate tunnels from all sources to all destinations that can significantly increase the size of the virtual layer. This also explains the deviation of marked values (*). Since the cost of the internal links of a virtual tunnel is always set to low by the heuristic, it can “re-use” those by connecting unprotected source nodes at later iteration steps. In contrast, the ILP uses higher link costs to maximize the level of protection at each step that sometimes leads to a local optimum.

The logarithmic progression of the algorithms is clearly visible on Figs. 3.6-3.7. In the first phase there is a steep increase in the LFA protection and the performance gap between the algorithms is minimal. We also observe that in most steps the algorithms prefer to add a single virtual node, however there are cases (see e.g., Fig. 3.6b) when both methods need a tunnel to overcome a certain complex scenario. We also show a topology (Deltacom) where the ILP with $k = 3$ does not perform in acceptable running time; this validates the need for efficient heuristics. In this special case we relaxed $k = 1$ for the ILP and kept $k = 3$ for the heuristic that revealed a 2-3% gain



(a) Germany_50 (b) Deltacom

Figure 3.7: Progression of LFA coverage in backbone topologies.

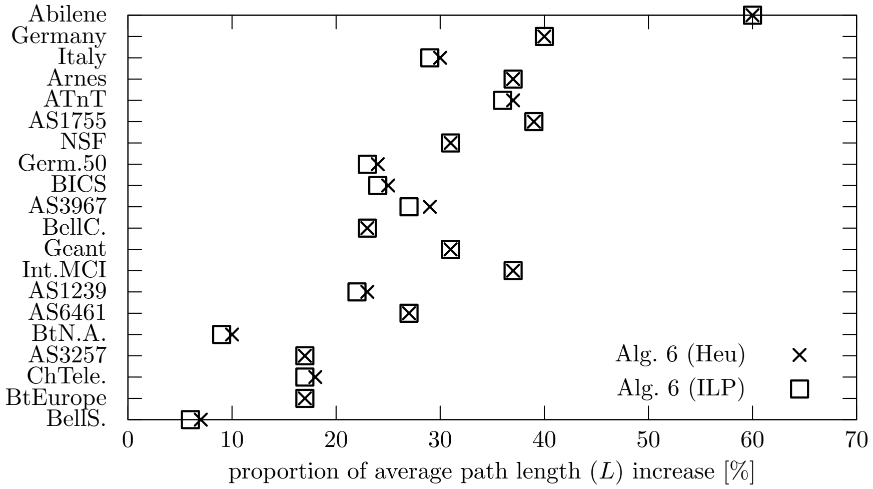


Figure 3.8: Average path stretch for single link failures.

on the ILP side in the first phase, but it got stuck after all, while the heuristic was still able to improve the coverage, see Fig. 3.7b.

Finally, we evaluate the change in the length of the detours in order to demonstrate that the traffic entering the virtual layer does not spend too much time there, and hence physical resources are not used extensively due to a failure. The values provided in Fig. 3.8 are the overhead compared to the default shortest path with the interpretation that in case of a failure, packets take in average $\sim$ 30% longer paths than usual. The results reveal negligible difference between the performance of the heuristic and the optimal version of GLFAVirt.

As a summary, we can conclude that our heuristic performs very close to the ILP. Both algorithms can bring small and middle-sized networks close to perfect LFA-coverage by provisioning just a couple of virtual routers, and in larger backbones we see similar improved protection coverage just by provisioning roughly one virtual router per node on average.

3.4 Application of Results

To our best knowledge, we are the first who showed that reaching perfect LFA coverage is achievable by creating a virtual overlay that does not interfere with the original shortest paths at all. Based on our results, Tapolcai showed a construction in [67] that achieves full LFA coverage by adding limited number of virtual nodes. The idea lies in creating two pairs of redundant trees in the virtual layer that one way or another serve as an alternate path from each source to each destination upon a link failure. Furthermore, it is shown that in case of 4-connected graphs a single pair of redundant trees is enough to achieve perfect coverage.

Our results appear in the IEEE survey mentioned before [57] (see Sec. 2.4) in the context of *Virtual Routing Overlays* that is concluded to finally solve multiple problems of inherent failure isolation, such as how to pin a packet into an overlay until it reaches its destination.

3.5 Related Work

In addition to the studies we overviewed in Sec. 2.5, we must mention “Fibbing”, a concept presented by Vissicchio *et al.* [65], whereby routers are tricked by using virtual routers into sending traffic to the desired direction. The idea is to mix the advantages of centralized control (e.g. Software Defined Networking (SDN)), such as fast configuration, with the robustness of the legacy distributed routing protocols (e.g. OSPF). However, Fibbing requires the presence of a centralized SDN controller, whereas our proposal remains completely within the conventional distributed IP routing model.

With respect to the efforts building on network optimization to increase LFA coverage, we refer to [68] that came up with the idea of creating virtual routers for otherwise unprotected node pairs. The authors try to answer the question of how to build a minimal overlay that results in perfect LFA coverage, but they found the problem, even in a very minimalistic setting, NP-complete. The initial results suggest that building an entire overlay in one step is a too ambitious goal, and they also leave the question open about how to properly create such an overlay.

Chapter 4

R3D3: A Doubly Opportunistic Data Structure

After concluding our efforts on making communication networks more reliable, we now move to the field of data compression in order to enhance the performance of computing devices. First, we briefly go through the history and evolution of information theory and the basics of data compression. We discuss what succinct data structures are, how do they arise and why they are becoming essential in computer science.

4.1 Background

In the beginning of the 20th century it was already possible to transmit messages over wires or radio paths, but researchers were still struggling to quantify and measure the base building block of communication — the *information.* In 1928, the goal of *Ralph Hartley* was to define a quantitative measure to be able to compare the capacity of different systems like telegraphy, telephony or picture transmission [69]. He found the information to be a "very elastic term", but he already discovered that for being able to uniquely encode a string of length n on an alphabet of size δ requires $n \log(\delta)$ "unit of information".

Twenty years later, by publishing "A Mathematical Theory of Communication" [70], *Claude Shannon* laid the foundation for the field of information theory. He introduced an absolute precise unit of information, called *bit*, that can be used to encode all kinds of media like telephone signals, text, radio waves, pictures ...etc. Not surprisingly, this paper is often referred to the beginning of the digital age. His work revealed that the message itself is irrelevant to the media to be transmitted, regardless if it is a telegraph or a video message.

Aside of proposing a unified representation of data, Shannon also quantified the amount of information that a signal holds. Suppose that there is a set of possible events with the probabilities of occurrence $p_1, p_2, \ldots, p_n$. We only know the probabilities, but nothing about the upcoming event at a certain point of time. Now the question is how uncertain we are about the outcome, and how could we measure it? The answer is called *entropy* (H) that gives the amount of "useful" content of a message:

$$H = -\sum_{i=1}^{n} p_i \log(p_i) \tag{4.1}$$

Note that in case of two possible events (e.g. a bit is set or zero), Eq. 4.1 becomes:

$$H_0 = p\log(\frac{1}{p}) + (1-p)\log(\frac{1}{1-p}) \tag{4.2}$$

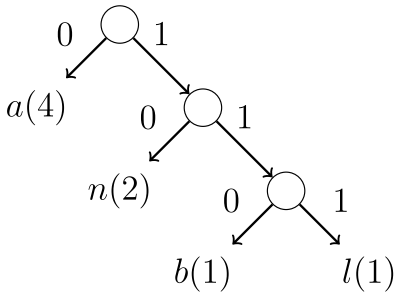
Observe that the expression evaluates to $H_0 = 0$ if either $p = 1$ or $p = 0$ with the interpretation that if we always know what to expect, then the uncertainty of the information source is zero. Similarly, if $p = 0.5$ then $H_0 = 1$, so we can hardly predict the next incoming symbol. Here, the notation H_0 refers to the case when there is no dependency between the symbols of a message, or if there is, then we do not take it into account. It is called the *zero-order entropy*. As opposite, sometimes it is possible to rely on the context before a symbol occurs that allows shorter representation, but throughout this paper we do not consider such cases.

Last but not least, Shannon also showed that entropy is the lower bound on the number of bits per symbol that is needed to encode a message without losing information. In other words, if there is a bitvector ($\delta = \{0, 1\}$) of length n and the occurrence of 1s is p, then no compression scheme can achieve better than nH_0 bits of space. Interestingly, the initial goal of compression was to speed up the transmission rate of radio channels by saving bandwidth, so the first *loss-less compressing* scheme that was proposed by Shannon and Fano was referred as "source coding". It was based on *variable length coding*, and despite of its novelty it is not used today because it is

symbol	*inf. th*	*Huff.*
a	00	0
b	01	110
l	10	111
n	11	10

l	a	b	a	n	a	n	a	$\sum bits$
10	00	01	00	11	00	11	00	16
111	0	110	0	10	0	10	0	14

(a) The code words in case of fix and variable length encoding.



(b) The Huffman-tree for the word "la-banana". The occurrences of each symbol are in parentheses.

Figure 4.1: A sample example for presenting the pros and cons of fixed and variable length encodings.

not guaranteed to terminate with the optimal result.

Three years later, *David A. Huffman*, a student of professor Fano, came up with a better encoding scheme that had been proven to perform closer to (or even achieve) the entropy. It is noteworthy that the base of today's popular file formats, like JPEG or PNG, are still built on variants of Huffman-coding. It uses variable length of code words as well, and sorts the symbols according to their probabilities: fewer bits for frequent symbols and more bits for rarely occurring ones. To optimize the encoding scheme, and to avoid ambiguity between codes of symbols, Huffman-coding implements a linear time greedy algorithm that builds a prefix tree for mapping all the symbols to the proper code words.

Let us recap these concepts through an example (Fig. 4.1). Suppose that our goal is to obtain the binary representation of the word "la banana" with the fewest bits possible. The alphabet is $\delta = \{a, b, l, n\}$, and the length of the string $n = 8$. As Hartley observed first, we only need 2 bits to represent 4 symbols, thus the final length of the encoded string is 16 bits. Thanks to the fixed length encoding, it provides easy access to the symbols at any positions. For instance to query the ith position, one just need to read 2 bits starting from the $2(i-1)$ position. On the other hand, we have not evaluated the space consumption against the entropy yet. The probabilities are $p_a = 1/2$, $p_b = 1/8$, $p_l = 1/8$ and $p_n = 1/4$, so $nH_0 = 14$ bits, revealing that this representation is not particularly memory efficient. Therefore, in the next round, we turn to Huffman-coding. The prefix tree is sorted by symbol frequencies that makes it possible to encode popular symbols on fewer bits (see Fig. 4.1b). One can easily verify that the length of the resulting bitvector is indeed equal to the entropy. But how can we access a particular string position now? Well, unfortunately we do not have better option than to decompress the string, and do the query on the original form.

If the "either-or" nature of this game had not become obvious yet, we explicitly highlight it: **either we store the data on memory efficient space, or we execute fast operations on it**. Consequently, the best performance never comes with maximal compression. This problem is often called the *space-time trade-off.*

But what would be the benefit of storing the information in a compressed and quickly accessible form? Thanks to the skyrocketing volume of electronic data, the information to be processed has greatly surpassed the increase in memory, disk and link capacities of today's computer systems [71, 72]. As a consequence, *it became crucial to be able to keep frequently used data structures close to the CPU, in the lower cache levels of the cache hierarchy, and at the same time to support fast operations on them.*

The field of succinct data structures was born from this concept since a group of researchers were not willing to accept the space-time compromise. This finally led to the appearance of a wide set of novel and versatile data structures that quickly became popular in computer science. Accordingly, **the last goal of the Dissertation is to present a novel succinct data structure that achieves better compression and/or execution speed compared to the existing methods**.

4.1.1 Succinct Data Structures

In general, we call a data structure *succinct*, if it can be stored on the optimal number of bits plus some little "extra space": $opt + o(opt)$. A succinct encoding of a bitmap t of length n is worst-case minimum $n + o(n)$ bits, and it also implements `rank` and `select` queries in $O(1)$ time. But what are these operations and why do they matter?

In [73] the goal of Jacobson was to find a more space efficient representation for static unlabeled trees and graphs compared to the conventional pointer based

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0

Figure 4.2: A sample bitvector.

approach. He argued that even though linked data structures offer rapid traversal, the space consumed by the "fat" pointers does often outweight the data itself. At that time, on the other hand, the operations on compressed trees and graphs (i.e. tree search) required $O(n)$ steps that severely hindered the applicability of such structures. Jacobson showed a new scheme that not only consumes minimal space, but also made it possible to do the tree traversal in constant time. His idea was to encode the tree into a bitvector, where the traversal from a node to its child (and to its parents respectively) can be expressed via constant number of `rank` and `select` operations. Today, these queries are *the basic functions of succinct representations* and they are applied to many other problem domains as well, such as text-indexing or DNA sequencing (detailed later) [74]. We define those operations on a bitvector t as follows:

- $\texttt{rank}_q(t, i)$: return the number of occurrences of symbol q in $t[1, i]$;
- $\texttt{select}_q(t, i)$: return the position of the i-th occurrence of symbol q in t.

Consider the example on Fig. 4.2. Here $\texttt{rank}_1(t, 8) = 2$ gives the number of bits set to 1 up to and counting the 8-th position, and $\texttt{select}_1(t, 2) = 7$ indicates that the second set bit occurs at position 7. Notice that `rank` and `select` are "dual" in that if $\texttt{select}_1(t, i) = m$ then $\texttt{rank}_1(t, m) = i$. Further, $\texttt{rank}_0(t, i) + \texttt{rank}_1(t, i) = i$ but the same does not hold for `select`.

To be able to answer those queries in constant time, Jacobson built an auxiliary data structure, a two-level dictionary, that was stored along the encoded bitvector. It consumed $o(n)$ bits and gave answer to `rank` in $O(1)$, and to `select` in $O(\log\log(n))$ times.

In 2005 Miltersen proved that the lower bound for storing `rank` dictionary is $\Omega(\frac{n \log\log n}{\log n})$ bits [75], and Golynski showed that this space is optimal if we store the bitmap in plain form and we build the index on top of it [76]. Similarly, the result for `select` was improved to $O(1)$ by Clark [77], but it still required $o(n)$ bits. In [76] Golynski showed that the lower bound for `select` dictionaries is $\Omega(\frac{n \log\log n}{\log n})$ too. Note that we call these techniques, when the input is stored in plain form without taking advantage of the possible compressibility of data, *systematic encoding* [78].

Aside of encoding trees and graphs, there were other problems in computer science that brought the attention to succinct data structures. The goal of *text indexing* is to count and locate the occurrences of arbitrary patterns in an input string. However, the exponential growth of electronic data makes it impossible to work with conventional data structures without suffering major performance drop. In 2000 Ferragina and Manzini introduced the concept of *opportunistic data structures* in order to take advantage of the potential compressibility of the input to decrease the space occupancy beyond the worst-case limit, at no significant slowdown in query performance [79]. Others were dealing with the *dictionary* problem that is for a given S set of distinct n keys is to find a proper representation of S, so that membership questions like "is $j \in S$" can be answered quickly. Here the two commonly used schemes, sorted arrays and hash maps, offer distinct benefits. While sorted arrays support constant time `rank`, they are slower in answering the membership questions comparing to hash maps [80].

The authors of [81], Raman, Raman and Rao, by building on the results of Brodnik and Munro [82], and Pagh [83], developed a *fully indexable dictionary (FID)* that combines the speed of hashing with the versatility of sorted arrays. It is a *compressed bitvector* representation that stores a bitmap t of length n on nH_0 bits, plus the index on $O(\frac{n \log\log n}{\log n}) = o(n)$ bits and implements `access`, `rank` and `select` queries in constant time. The literature refers it *RRR*, and it quickly became the prevalent bitmap compression scheme, used by the majority of succinct applications. In 2008 Pătraşcu showed that the space can be squeezed up to $nH_0 + O(n\frac{n}{\log^c(n)})$ bits, with a speed of `rank` and `select` queries of $O(c)$ time, for any constant c, and it is essentially optimal [84]. Again, if the encoded bitvector is stored in a compressed form, then we call the encoding scheme *non-systematic*, and since it offers the same performance with better space utilization, they are in the main focus of the research activities around information retrieval and data compression [78].

4.1.2 Wavelet Tree

Undoubtedly, there is a wide range of applications that benefit from compressed bitvectors, but we limit our discussion to the most popular one, called the wavelet

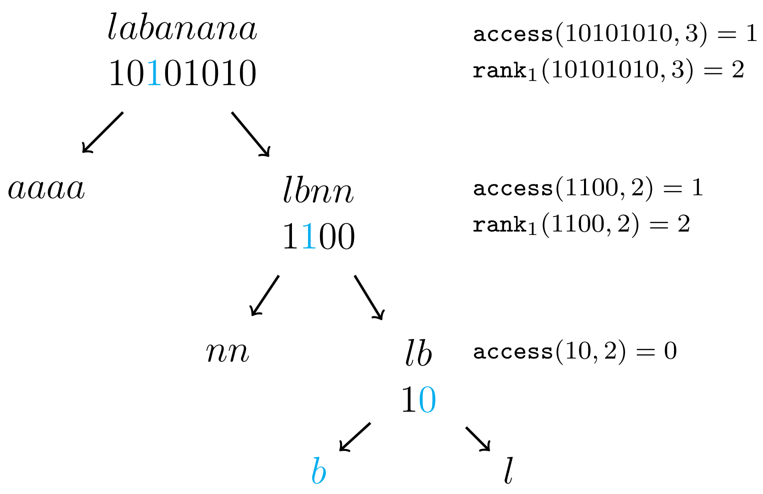


Figure 4.3: A Huffman shaped wavelet tree.

tree. It was developed by Grossi, Gupta and Vitter [85] in 2003, and it immediately became a milestone in text indexing and computational geometry [86]. Since then, dozens of applications appear every year that are built on wavelet trees and offer elegant solutions to a wide variety of problems (see later). We now overview the main principles in order to point out how bitmap compression, one of the main objectives of the Dissertation, integrates into the wavelet tree model. Besides, our objective is to give intuition into the simple but beautiful idea that made this data structure essential in the succinct world.

Originally, the wavelet tree is a balanced binary tree for a sequence $s[1 \ldots n]$ over an alphabet σ. The leaves of the tree form a hierarchy of bitvectors, where t_{leaf} is defined as follows: if $s[i] \leq \sigma/2$, then $t_{leaf}[i] = 0$ else $t_{leaf}[i] = 1$. By recursively applying this step we get a tree of depth $\log(\sigma)$. The upper bound for the space consumption is $n \log(\sigma)$. In [87] Mäkinen and Navarro proposed to give Huffman shape to the wavelet tree, where each symbol is encoded according to its frequency of appearance in s. This improves the overall size of the tree to $n(H_0(s) + 1)$ bits.

Such a Huffman shaped wavelet tree is depicted on Fig. 4.3. Assume that our goal is to answer the query `access`(3) on the encoded word "labanana". First, the access operation on the top level bitvector defines which branch to take in the tree. In our case the $3rd$ bit is set, so we know that we proceed on the right branch. To figure out the index of the symbol in the child node, we have to count the number of preceding symbols of the given branch up to ours that is the exact definition of $\texttt{rank}_{branch}(i)$. Accordingly, $\texttt{rank}_1(3)$ gives back the index of b in the child node (2) from where we can recursively continue the same operations until the bottom of the tree is reached. The very bottom node returns the answer: b. Note that only bitmaps are stored at each level of the tree and the strings are merely present for demonstration purposes.

A very important observation is that **the size of the bitmaps, and consequently the size of the wavelet tree, can be compressed** if we still support constant time `access` and `rank` operations. Recall that RRR [81] is a piece that perfectly fits into this puzzle, making it the base building block of wavelet trees.

4.1.3 Bitmap Compression Schemes

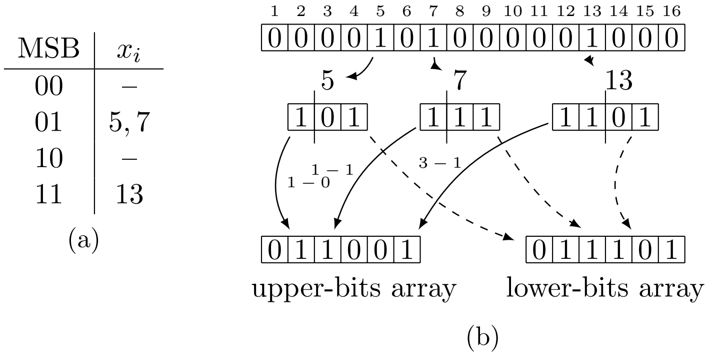
Now, as we have noticed in the previous section, bitmaps are the fundamental building blocks of the succinct representation. In this section we delve deeper into the most popular bitmap compression methods.

The Elias–Fano coding scheme

Although the scheme proposed by Elias in [88] is not considered to be succinct, but is often called *quasi-succinct* as the compressed size is very close to optimal (less than half a bit per element)[89]. We, however, briefly overview this technique not just because it is fundamental in data compression studies, but also because the Dissertation heavily builds on this coding scheme. Note that practical implementations of the plain form are very inefficient, therefore we rather show an optimized version in the upcoming section (SD-array).

Definition 13. *For a given bitmap t of length n, we call the number of set bits in t, i.e. $\texttt{rank}_1(n)$, the population count and we denote it with $\texttt{popcount}(t)$.*

The Elias–Fano coding scheme stores a bitvector t in $nH_0 + O(n)$ space and answers $\texttt{select}_1$ queries in $O(1)$ time, with no support for `rank` and `access`. Herein, we describe an alternative scheme *EF* that attains $nH_0 + O(m)$ bits of space and needs $O(m)$ time for `access`, `rank`, and `select`, where $m = \texttt{popcount}(t)$ (see also [81, 90, 91]).



MSB	x_i
00	–
01	5, 7
10	–
11	13

(a)

(b)

Figure 4.4: Elias–Fano encoding scheme: (a) MSB bucketing on the characteristic vector $(5, 7, 13)$, and (b) EF encoding.

The EF scheme encodes the characteristic vector $\{x_1, x_2, \ldots, x_m\}$ of t, where $m =$ $\texttt{popcount}(t)$ and $x_i = \texttt{select}_1(t, i) : i \in \{1, \ldots, m\}$, instead of t itself, using a technique called MSB bucketing: group x_is according to the most significant $\log m$ bits into buckets, store the $l = \log n - \log m = \log \frac{n}{m}$ lower-order bits for each x_i verbatim in an array (called the Lower-bits Array, *LBA*), and store the significant bits as a sequence of unary encoded gaps in another array (the Upper-bits Array, *UBA*). The UBA is constructed as follows: for each bucket write down as many 1s as there are x_is in the bucket followed by a 0.

Perhaps an example is in order here. In Fig. 4.4, $x_1 = 5$, $x_2 = 7$ and $x_3 = 13$, $n = 16$ and $m = 3$, so $l = \lfloor \log {}^{16}/_{3} \rfloor = 2$. Thus, the LBA will contain the lower $l = 2$ bits of each x_i verbatim. Further, the bucket size is $2^l = 4$, and so the number of x_is in each bucket is $0, 2, 0, 1$, whose unary encoding gives the UBA: 0110010 (the last 0 can be omitted).

Storing the m elements of the LBA takes $m \log \frac{n}{m}$ bits while the UBA needs $2^{\log m} + m = O(m)$ bits, as there are as many 0s as there are buckets plus m bits set to 1 for each x_i in $i \in \{1, \ldots, m\}$. Finally, we need an additional $\log m$ bits to store m, which we omit here for reasons that will be made clear later. The overall size of EF is $m \log \frac{n}{m} + 2^{\log m} + m = nH_0 + O(m)$ bits, where the data component (the LBA) takes nH_0 bits and the index (the UBA) takes another $O(m)$.

Now, answering $\texttt{access}(t, i)$ goes as follows. First, find the bucket q that contains position i: $q = \frac{i}{2^l}$, then we find the run of 1s in the UBA that corresponds to the q-th bucket: $z = \texttt{select}_0(\mathit{UBA}, q)$; we observe that there were exactly $z - q$ occurrences of 1s in the UBA *before* position z so we scan the LBA *leftward* from position $(z-q)l$, decoding at most 2^l elements x_j of the characteristic vector; if for some $x_j = i$, then the result of the query is 1, otherwise 0. This takes $O(m)$ steps in general.

SD-array

Okanohara and Sadakane introduced a novel data structure, the *sdarray* [90]. The idea is to differentiate sparse and dense sets and so they treat them separately. They use the Elias–Fano scheme [88] for encoding the set bit positions in the input bitmap, coupled with the index proposed by [77], to achieve very fast `rank` and `select` queries. Unfortunately, the density of the input heavily influences the size of the index structure, which can easily become prohibitive in practice (i.e., an extra 50% space toll is paid if we need both `rank` and `select` at the same time).

The authors of [92] showed that a refined RRR implementation achieves better compression than *sdarray* (and the other structures of [90]) if the bitmap density is around 20%.

RRR

We already touched upon the compressed bitvector representation of Raman, Raman and Rao (RRR) [81]. RRR comprises a *block-coding component* to encode the useful data and an indexing scheme to support queries to the blocks. The structure partitions t into blocks $b_1, b_2, \ldots$ of size $b = \frac{\log n}{2}$ bits (see Fig. 4.5 for an illustration). Each block b_i is encoded with a pair (c_i, o_i), where $c_i = \texttt{popcount}(b_i)$ is the *class* of b_i and o_i is the offset that is used as an index to a prefab table. The global universal table, for a given class c, translates any offset into the proper bit vector b.

Thanks to the sophisticated two-level superblock/block *indexing scheme* RRR supports $O(1)$ time `access`, `rank` and `select`. The idea is to group every consecutive $\log n$ blocks into a superblock. The superblock explicitly contains the cumulative rank up to the superblock's beginning, and it also maintains a relative index to the starting positions of the blocks together with their cumulative ranks.

Answering $\texttt{rank}(t, i)$ works as follows. As blocks and superblocks span constant number of bits in t, i uniquely determines the block and the superblock that contains position i. We follow first the superblock pointer and initialize the rank answer to

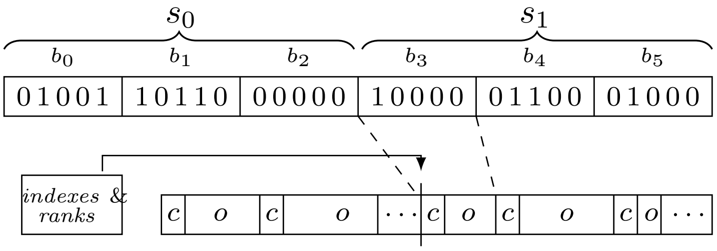


Figure 4.5: Sketch of the *RRR* encoding scheme.

the cumulative rank, as stored in the index, until the superblock's starting position. We do the same step within the superblock now by reading the superblock specific relative rank that corresponds to the block's starting position, and we also obtain the offset. By the help of the offset, we get back the bitmap and the block specific rank answer from the universal table.

Unfortunately the size of the universal table grows exponentially by the size of the blocks, as there are 2^b variations on b bits that adds up to $\frac{s}{b}2^b$ bits in total. On the other hand, if the size of the blocks remain small, their number will increase that requires more space for the indexing. Therefore, Claude and Navarro proposed an optimized RRR in [92] and [93], which attains significant space reduction by substituting RRR's universal block coding tables with *on-the-fly block decoding using combinatorial unranking* [94], and removing the relative block pointers /rank counters from the index and resorting to linear search inside superblocks. They achieved about 50% space reduction at the price of worsening the query performance from $O(1)$ to $O(b)$. The implementation of this RRR version, thanks to its efficiency, became very popular [95] and therefore, from now on, *we also refer to the optimized version if we talk about RRR.*

Answering $\texttt{rank}(t, i)$ goes very similar on the improved RRR, except that for finding the desired block we execute a linear search inside the superblock, and during that we accumulate the rank values (the classes) from each block along the way into our rank answer. Since the size of each block offset depends on its block class, the linear search uses a pre-computed dictionary that returns the offset size $\lceil \log \binom{b}{c} \rceil$ for each class c. This time the offset of the block contains a combinatorial rank that is defined as the sequence number of b_i in some fixed enumeration (e.g., lexicographic order) of all combinations of exactly c_i occurrences of 1s at b bit positions. Decoding the block (the so-called *combinatorial unranking* operation) and block-rank queries need $O(b) = O(\log n)$ time. Practical implementations have shown that basically any setting beyond $b = 63$ makes combinatorial unranking for block-decoding painfully slow [94, 96].

4.2 New Results

In this chapter, we introduce a novel data structure for *succinct bitmap compression* that is intended to be a competitive alternative to the best known encoding schemes, like the RRR.

4.2.1 Motivation

Supporting quick queries on compressed data requires indexing. Despite holding only a map to navigate on the compressed form of the useful data, it consumes noticeable space. As the size of the data grows so does the index and there is a point when **it outgrows the cache that leads to painfully slow queries due to the expensive memory or disk access**. There are various practical consequences of this phenomena starting from slower web searching to poorly performing networking devices that are not able to keep up with the growth of forwarding tables anymore [97] and hence they rather store the routable prefixes in a compressed form [98].

The problem with current bitmap compression methods is that **they resort to compress the data only without addressing to reduce the size of the index**. One way to solve this problem is to use larger block sizes that implicitly leads to store fewer indexes. This indeed reduces the overall size of RRR, but unfortunately at the price of a drastic slow down in queries caused by the expensive combinatorial unranking of large blocks.

Our goal is to present a bitmap compression scheme, called R3D3 ("RRR–Developed Data structure for big Data"), that manages to reduce the size of the index by using larger blocks without introducing major performance drop in the speed of the queries compared to the case of small blocks. We call it a *doubly opportunistic data structure*, which, as opposed to conventional (singly) opportunistic schemes, **achieves entropy-constrained space on the data and the index at the same time** and

Table 4.1: Space–time complexity of bitvector compression methods, for bitmaps of length n, popcount m, block size b, and zero-order empirical entropy H_0.

			Speed	
	Method	Size (bits)	Rank	Select
Syst.	Uncompressed	n	$O(n)$	$O(n)$
	Miltersen, `rank` [75]	$\Omega(\frac{n \log\log n}{\log n})$	$O(1)$	-
	Golynski, `select` [76]	$\Omega(\frac{n \log\log n}{\log n})$	-	$O(1)$
Non-syst.	Pătraşcu [84]	$nH_0 + O(n\frac{n}{\log^c(n)})$	$O(c)$	$O(c)$
	sdarray [90]	$m \log \frac{n}{m} + 2m + o(m)$	$O(\log \frac{n}{m})+O(\frac{\log^4 m}{\log n})$	$O(\frac{\log^4 m}{\log n})$
	RRR [81]	$nH_0 + O(\frac{n \log\log n}{\log n})$	$O(1)$	$O(1)$
	RRR (optimized)[93]	$nH_0 + o(n)$	$O(\log n)$	$O(\log n)$
	R3D3	$nH_0 + nH_0(1/2 + O(\frac{\log\log n}{\log n}))$	$O(\log n)$	$O(\log n)$

allows random `access`, `rank` and `select` queries in $O(\log n)$ time.

Finally, we give a short overview on the collection of theoretical results and their practical implementations in Table 4.1. Notice the last entry, R3D3, that is our proposed data structure.

4.2.2 R3D3

R3D3 combines the storage scheme of RRR [93] with the Elias–Fano encoding scheme to create a very efficient bitmap encoding. The idea is to rely on RRR's indexing scheme, as it gives very fast access to block-codes and block-ranks, whereas the block-coding component will be changed from combinatorial unranking to the EF scheme.

The RRR index size is chiefly shaped by the block size b; the larger the block size the fewer blocks we need, and hence the fewer the costly block pointers and block-ranks. However, as we already pointed out, increasing the block size of RRR does not come for free, as the `access` and `rank` execution times are dominated by the block-coding component's running time $O(b)$. Since with combinatorial unranking a block is encoded with the same amount of steps regardless of the content of the block itself, our proposal is to swap this encoding scheme with something else that only considers the density of a given block. The EF coding described in Sec. 4.1.3 has such property: *EF's efficiency depends fundamentally on the number of* 1*s in a block and not the block size itself.*

Decoding a block b_i requires only $O(c_i)$ steps with EF where, recall, c_i is the class

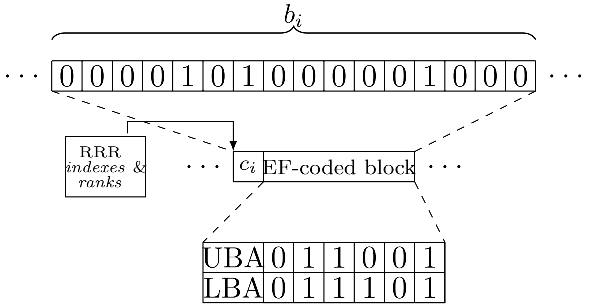


Figure 4.6: R3D3 encoding, with a single 16-bit block and the corresponding EF block-code. The superblock pointers and block classes are encoded in the RRR index, while the blocks are encoded with EF.

of b_i: $c_i = \texttt{popcount}(b_i)$, in contrast to the $O(b)$ time complexity of combinatorial unranking. Hence, we can safely increase the block size b to save space on RRR's indexing until we reach block-coding execution-time parity with RRR, which occurs when $\texttt{popcount}(b_i) = O(\log n)$. At this point our larger blocks will contain as many 1s as the default block size $O(\log n)$ of RRR and so both will need $O(\log n)$ steps for block-decoding, but we gain significant space on the indexing, thanks to the large blocks. The basic structure of R3D3 is given in Fig. 4.6.

Note, that our EF-coded blocks will be slightly larger than in RRR, but on the one hand this overhead will be low for small entropy input and, on the other hand, the significant performance gain in block-decoding will greatly compensate for this loss, as we demonstrate in our empirical studies in Section 4.2.3 and 4.2.4.

Building R3D3 goes very similarly to how it happens with RRR, just the block-coder is now EF instead of combinatorial ranking/unranking. First, we divide the input t into superblocks of size s and blocks of size b (we set these parameters later), build the RRR index, encode the class c_i for each block b_i directly and then invoke EF to encode b_i. Note that the input to EF is now the block b_i and the length equals b. To control c_i and get better compression we do the usual trick that if $\texttt{popcount}(n) > \frac{n}{2}$ then we encode the inverse of t instead of t. We also need to update the pre-computed dictionary used for navigating within a superblock, because the length of EF-encoded blocks is different.

R3D3 adopts a scheme we call *duplicate indexing*; it first invokes the RRR indexes to find the starting position for each block and then looks up the UBA to index the relevant entries in the LBA, and finally only a few LB entries need to be directly decoded. As the below analysis reveals, duplicate indexing yields a highly space- and time-efficient compressed bitvector data structure.

4.2.3 Space-Time Analysis

We set the superblock size to the same that is used by the RRR; the block size b will be determined later. With this parameter setting, the result below gives the storage space and the query times for R3D3.

Theorem 8. *Let t be a bitvector of length n, let $p = {}^{popcount(t)}/_{n}$, let H_0 be the zero-order empirical entropy of t, and fix the superblock size at $s = b \log n$ and block size at b. Then, encoding t with R3D3 needs at most*

$$nH_0 + np + \frac{n}{b}\,(2 + \log b) \quad \textit{bits.}$$

Proof. First, the index structure needs $\frac{n}{b}\,(2 + \log b)$ bits: it stores the address and the cumulative rank for each superblock, both requiring $\frac{n}{s}\,(\log n)$ bits space, plus the block-class table consuming another $\frac{n}{b}\,(\log b)$ bits. By substituting $s = b \log n$ we get the result.

Next, we show that the EF-encoded data adds up to $nH_0 + np$ bits; from this, nH_0 bits constitute the data portion (the LBA) and the other np bits count to the index (the UBA) as per duplicate indexing.

We observe that instead of calculating the space occupancy of each block b_i one by one it is enough to deal with the size of an "average" block where the class of the block can be expressed as $c = pb$ (the proof is trivial using Jensen's inequality, we omit the details). The UBA stores one bit for each bucket plus another bit for each bit set in the block, yielding $\frac{b}{2^l} + c$ bits overall, while the LBA consists of c elements, each of l bits. Summed up for each of the $\frac{n}{b}$ blocks we get $\frac{n}{b}\left(\frac{b}{2^l} + c + lc\right) = n\left(2^{-l} + p + pl\right)$. Recall that the choice for parameter l is elemental in EF; usually $l = \lfloor \log \frac{b}{c} \rfloor$.

First, we show the procession of the proof for the simplified case when we omit rounding to integers and let $l = \log b - \log c$, then we discuss how to handle this discrepancy. Letting $l = \log b - \log c$ firstly yields

$$\begin{aligned} &n\left(\tfrac{c}{b} + p + p \log \tfrac{b}{c}\right) = n\left(p + p + p \log \tfrac{1}{p}\right) \\ &\leq n\left(p + (1-p) \log \tfrac{1}{1-p} + p \log \tfrac{1}{p}\right) = np + nH_0 \ , \end{aligned} \tag{4.3}$$

by that $p \leq (1-p)\log(\frac{1}{1-p})$ if $p \in (0, \frac{1}{2}]$, as requested.

Now, to take care of integrality in $l = \lfloor \log \frac{b}{c} \rfloor$, using $\frac{b}{c} = \frac{1}{p}$ we write the data component size as $n\left(2^{-\lfloor \log \frac{1}{p} \rfloor} + p + p\left\lfloor \log \frac{1}{p} \right\rfloor\right)$. To prove the statement, it is now enough to show that this size is smaller than (4.3) or, equivalently, that for the difference

$$n\left(p + p + p\log\tfrac{1}{p} - \left(2^{-\lfloor \log \frac{1}{p} \rfloor} + p + p\left\lfloor \log \tfrac{1}{p} \right\rfloor\right)\right) \geq 0 \ . \tag{4.4}$$

Divide by $n > 0$ and simplify to get $p + p\log\frac{1}{p} - \left(2^{-\lfloor \log \frac{1}{p} \rfloor} + p\left\lfloor \log \frac{1}{p} \right\rfloor\right)$. Next, substitute $p = 2^{-i-r}$, where i is the integer part and $r = [0,1)$ is the remaining fractional part. Note that $i + r = \log\frac{1}{p}$ and $i = \lfloor \log \frac{1}{p} \rfloor$. This leads to $2^{-i-r} + 2^{-i-r}(i + r) - (2^{-i} + 2^{-i-r}i) = 2^{-i}\left(2^{-r} + 2^{-r}i + 2^{-r}r - 1 - 2^{-r}i\right) = 2^{-i}\left(2^{-r} + 2^{-r}r - 1\right) = 2^{-i}\left(2^{-r}(1+r) - 1\right)$. Since $2^{-i} > 0$ we need to show that $2^{-r}(1+r) \geq 1$, which can be written as $1 + r \geq 2^r$, where $r = [0,1)$. Noting that, 2^r is a convex function intersecting the line $1 + r$ at points $r = 0$ and $r = 1$ completes the proof. □

The query execution times for R3D3 are as follows: the time complexity for `access`(t,i) is dominated by the linear search to locate the beginning of the EF-coded block containing position i and identifying the class, which, similarly to RRR, is constrained by $O(\log n)$, to which block-decoding gives another $O(pb)$ steps. To reach parity with RRR, we can choose much larger block sizes (recall, in RRR the block size is $b = O(\log n)$, while in R3D3 $pb = O(\log n)$ and $p < 1$), which brings substantial space reduction as we need to store fewer block class values in the index (see below in Theorem 8). The same holds for `rank`(t,i). As per `select`(t,i), first we binary-search superblock and block ranks in $O(\log n)$ time and then decode the block, again in expected $O(pb)$ time. The total time for these queries is $O(\log n) + O(pb) = O(\log n)$ if $pb = O(\log n)$. We summarize these findings as follows.

Theorem 9. *Let t be a bitvector of length n and entropy H_0. Then, encoding t with R3D3 needs at most*

$$nH_0 + nH_0\left(\frac{1}{2} + O\left(\frac{\log\log n}{\log n}\right)\right) \tag{4.5}$$

bits and supports `access`, `rank`, and `select` in expected $O(\log n)$ time.

Proof. We choose $pb = O(\log n)$ and use Theorem 8. Let us denote $pb = C$ and substitute C in $\frac{n}{b}(2 + \log b)$ to get $\frac{np}{C}(2 + \log\frac{C}{p})$. We know that $p \leq \frac{1}{2}$ and so

$p \log \frac{1}{p} \leq H_0$. Furthermore, $2p \leq H_0$, since function values at the interval borders ($p \in [0, \frac{1}{2}]$) coincide (0 and 1 respectively), and H_0 is concave in this range (since its second derivative is negative) while $2p$ is linear. Using $\frac{2np}{C} \leq \frac{nH_0}{C}$ and

$$\frac{np}{C} \log \frac{C}{p} = \frac{n}{C} \left(p \log \frac{1}{p} + p \log C \right) \leq nH_0 O \left(\frac{\log C}{C} \right).$$

By substituting the above and writing $np \leq \frac{1}{2} nH_0$ into Theorem 8 we get:

$$nH_0 + np + \frac{n}{b}(2 + \log b) \leq nH_0 + nH_0 \left(\frac{1}{2} + O\left(\frac{\log \log n}{\log n} \right) \right) \tag{4.6}$$

that completes the proof. □

In summary, R3D3 attains smaller encoding space than RRR, as the time-efficiency gain of EF block-coding over combinatorial unranking allows us to use larger blocks and hence store fewer block classes in the index, while it supports operations on the compressed data in the same time-complexity as RRR, $O(\log n)$. In our implementation (see the next section), however, we choose a different approach: we experimentally set the block size to reach encoding-size parity with RRR and we exploit the resultant running time gains of R3D3; below we use the block sizes $b = 128$ and $b = 256$ bits. This choice was made to support highly time-sensitive applications (e.g., [98]) more efficiently.

4.2.4 Experimental Results

Next, we turn to present a comprehensive set of experimental results to evaluate the space- and time-efficiency of R3D3. For this purpose, we created a proof-of-concept prototype on top of the Succinct Data Structure Library (SDSL, [95]) that is available at [99]. The two dimensions of interest are the compressed size and performance of queries for RRR and R3D3. As of our knowledge the most efficient RRR implementation is offered by SDSL, we tested our prototype against this version. We used the CPU's `TSC` register, holding the actual snapshot of the program counter, to measure with close-to cycle-level precision. The experiments were conducted on a Linux PC, Intel Core i3 CPU @ 3.3Ghz with 4Gbyte of RAM.

Block-coding. The goal of our first experiment is to validate our choice for EF

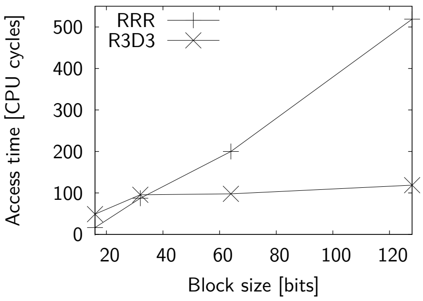


Figure 4.7: Average time to access a random position in RRR and EF block-codes as the function of the block size, on random bitmaps, $p = 0.1$.

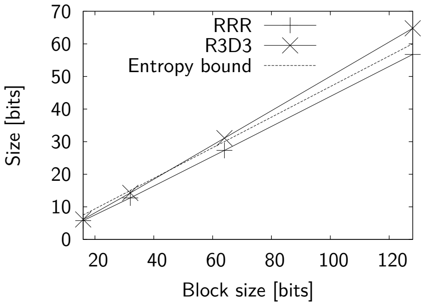


Figure 4.8: Average size of RRR and EF block-codes and the zero-order entropy limit as the function of the block size, on random bitmaps, $p = 0.1$.

instead of RRR's combinatorial ranking/unranking scheme to encode blocks. Recall, this choice was made because EF supports all basic block-operations in $O(\texttt{popcount}(b))$ time as opposed to $O(b)$ for RRR, where b is the block size, at the cost of slightly bigger block-codes.

We made 1 million trials, each time with a new random block generated by setting each bit to 1 independently with probability $p = 0.1$ (results were similar with different ps). Fig. 4.7 gives the average time to make a random access to the encoded blocks and Fig. 4.8 compares the average size of block-codes, as the block size increases from 16 to 128.

We observe that EF block-coding is indeed much less sensitive to the block size; while R3D3 needs only 3 times as much time to access a 128-bit block as for a 16-bit block, this factor is 25-fold with RRR. Furthermore, R3D3 produces only slightly larger blocks than RRR and both are comfortably close to the entropy bound (that RRR beats the entropy limit is not surprising, as combinatorial ranks are a maximally space-efficient universal code, plus our results do not account for the storage size of the class bits c_i).

Random synthetic bitmaps. For this experiment we again use random bitmaps as input, but now we evaluate RRR and R3D3 in their entirety (not just the block-coding component), the same way as in [93]. In particular, we measure the average execution time of `rank` and `select` queries on the compressed data structures against the total compressed encoding size. We used random bitmaps of the same length as [93] (2^{28} bits) and we set the same superblock sizes (32, 64 and 128); RRR was measured with

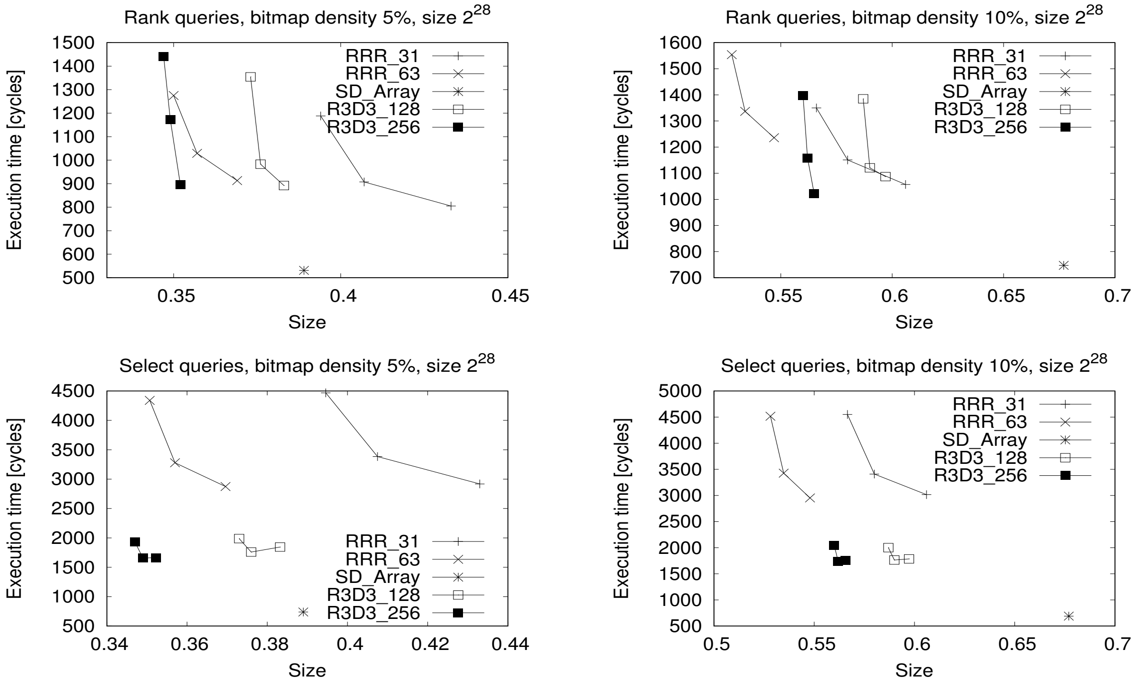


Figure 4.9: Average space–time efficiency of RRR and R3D3 on random rank and random select queries, on random 2^{28} bit long bitmaps.

block size 31 (RRR_31) and 63 (RRR_63), whereas for R3D3 we used block sizes 128 (R3D3_128) and 256 (R3D3_256). We also included the results for sdarrays (SD_array, [90]). The reported execution times are averages over 50,000 random queries. Fig. 4.9 shows for the results.

Our observations are as follows. At low density, R3D3_256 needs roughly 2-3% less space than RRR and attains similar performance in rank queries and up to 3 times better performance on select queries. In addition, R3D3 exhibits 6-8% better compression compared to the SD_Array. For a higher 10% density R3D3 achieves around 15-20% performance edge over RRR in rank queries and again multiple-times better execution times on select.

Complex data. Compressed bitmap indexes are the major building blocks for most complex data structures, like wavelet trees [85] and compressed Internet forwarding tables [98]. Correspondingly, we also studied how R3D3 performs when embedded into such complex data structures.

First, we experimented with textual data: we took different instances from the [100] (dna), [101] (chr7, chr22) and [102] (coli, pi) corpus and indexed them with

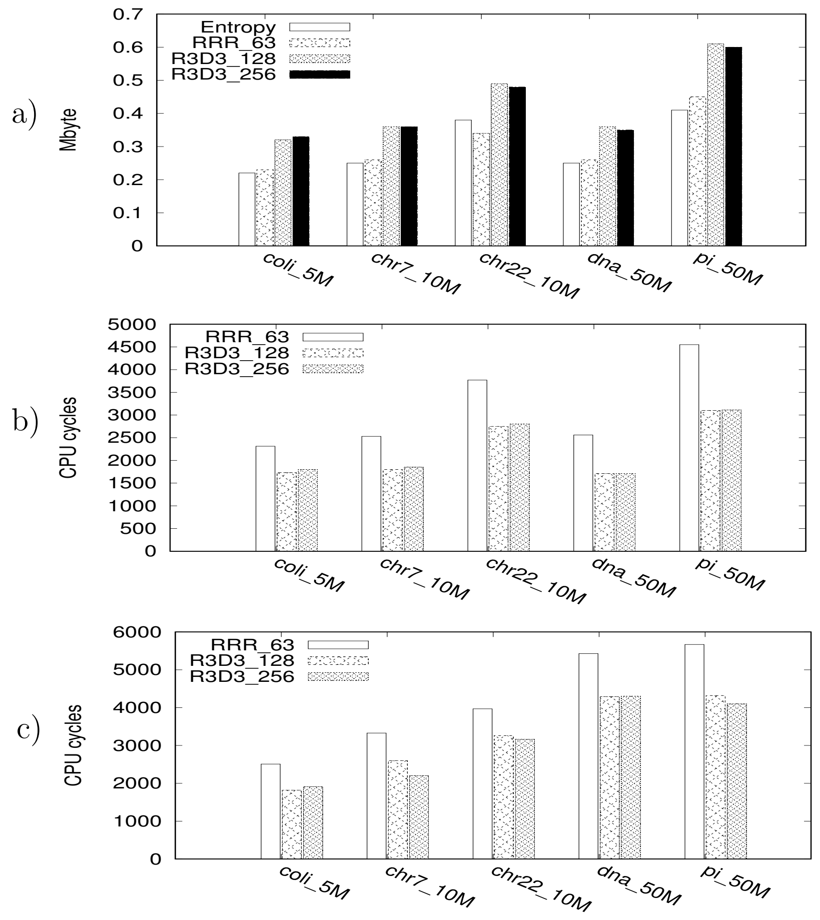


Figure 4.10: RRR and R3D3 in wavelet trees on a real corpus: (a) size [Mbytes] and execution time of (b) random `access` and (c) random `rank` queries [CPU-cycles].

Huffman-shaped wavelet trees. Here again, running time results are averages over $50,000$ random `access` and `rank` queries. The results in Fig. 4.10 reproduce our earlier findings: R3D3 outperforms RRR by 25-35% in `access` queries and roughly 20-26% in `rank`, at the cost of a moderate (up to 15%) increase in storage size.

These observations are confirmed by our experiments on real Internet forwarding tables, a relatively new application of complex data structures to encode routing information within the very limiting memory constraints of routers' embedded ASICs but still perform longest-prefix-matching lookups very fast [98]. The results for 3 real forwarding table instances, downloaded from operational Internet routers, are given

Table 4.2: RRR or R3D3 in forwarding table encoding: table sample name, number of prefixes, and entropy bound as of [98]; and compressed size and average execution time of random FIB lookups. Sizes are in Kbytes (KiB) and times in CPU cycles.

			RRR_63		R3D3_128		R3D3_256	
Name	#Prefixes	Entropy	Size	Lookup	Size	Lookup	Size	Lookup
`hbone-szeged`	453,685	70.1	63.1	21340	86.9	12670	93	11800
`access_d`	403,245	149.1	83.5	18670	115.4	11000	124	10900
`access_v`	2,970	1.08	4.6	20960	6.2	14800	6.5	11540
`mobile`	4,391	1.32	3.4	15400	3.7	10960	3.8	11580
`hbone-vh1`	453,741	222.6	160.8	20340	222	13850	238	13690

in Table 4.2. Here, entropy is the tree entropy as of [98] and the compressed encodings were obtained by the *XBW* data structure, instantiated over RRR- and R3D3-coded bitmaps with different block sizes. Again, we see a modest (up to 40%) size increase with R3D3, at the cost of close to twice the lookup performance.

Summary. The efficiency R3D3 provides in compression is clear for sparse input data sets. On the other hand, by increasing the density of the input this trend breaks down around $p = 0.2\%$. One reason for this phenomena is the "quasi-succinct" gap encoding representation used by EF [89], that expands faster for denser inputs than the index could be reduced. Besides, the speed of queries were slightly slowed down due to the poor CPU support for register operations requiring more than 64 bits, that eventually translated larger block operations into several bit shifting and registers changing maneuvers.

Our experiments show that most benefits already manifest themselves at the block size of 128 and 256 bits with R3D3 at the cost of a modest increase in encoding size. Unfortunately, limitations of our current implementation precluded us from increasing the block size further even though this is usually considered beneficial and it is essentially free with R3D3; fine-tuning our implementation to the AVX2 and the upcoming AVX-512 SIMD CPU instruction sets, which will allow to operate on blocks as large as 512 bits with similar efficiency as native 64-bit integers. With such optimization in place, we expect further space–time gains with R3D3 over existing compressed bitmaps schemes.

4.3 Application of Results

In this section we give an insight into the list of applications where R3D3 could show potential in the future.

Text-indexing: Wavelet trees are extensively used for maintaining *full-text indexes* that is a utility to count and locate any arbitrary patterns on a string. It enables fast search on large text document collections, and helps to speed up advanced queries (e.g. filtered search) in conventional databases [103, 104, 105]. Wavelet trees are also used to implement *inverted indexes*, the main building block of Web search engines (e.g. Google, Bing). Essentially, it keeps track of the list of positions where words of the document occur, to speed up the localization and relevance ranking of web pages [106]. A similar yet different problem is the *document retrieval* problem, where the task is to return the list of documents in which a specific word appears. The restricted version is called the *top-k document retrieval* problem that takes into consideration the relevance of each document, hence it is able to return a decreasing order of interesting answers. Readers interested how wavelet trees are tailored to this task are referred to [107] and [108].

Graphs and Trees: As it was already pointed out by Jacobson in [73], the succinct representation of graphs and trees does not sacrifice the speed of traversal. In [92] Claude and Navarro showed how to use wavelet trees to encode *directed and undirected graphs*, and in [109] they further improved the idea to be competitive with current *Web graph compression* methods [86].

Genome Compression: The exponential accumulation of full genome data drew the attention to space efficient encoding schemes in the field of computational biology too. Here, the task is to find and access individual and subsequent patterns on the sequence of chromosomes that constitute the DNA [110, 111]. Just like in the previous use-cases, the wavelet tree encodes the genome sequence and allows to execute fast queries on the compressed form [112].

Data Mining and Machine Learning: In [113] Raman points out that succinct data structures can be easily used by *big data* applications to allow scalable mining, if they provide the same API as conventional data stores. Thanks to the wide range of implementations [114, 95], it has never been easier to take these structures into use, opening up the avenue for new application specific optimization and heuristics in data mining. Besides, a study from the field of *machine learning* also refers to the wavelet trees as an alternative to learned indexes [115], and concludes that there is a potential in succinct data structures for further study.

Databases: The bedrock of modern cloud services is the high performance key-value stores. However, they still struggle with answering interactive queries on the values, not the keys. Therefore, the authors of [116] proposed Succinct, a distributed data store, that provides fast operations with low memory footprint with the help of wavelet trees. They concluded that Succinct consumes ten times less memory yet still operates with dramatically lower latency compared to the today's most popular cloud databases: MongoDB [117] and Cassandra [118].

FIB compression: Last but not least, it was recently shown in [98] that compressed bitvectors can be also used to construct a space-efficient representation of Internet router's forwarding tables (FIBs). The resultant compressed IP FIBs have been shown to squeeze the routing table of a contemporary IPv4 router, counting beyond 500,000 prefixes, to a mere 70–200 Kbytes of memory, while supporting wire-speed longest-prefix matching right on the compressed form. The authors designed a tool [119] to obtain historic information about the information-theoretical entropy of Internet routing in cooperation with several ASes, such as HBONE, the Hungarian academic backbone system. The goal of the project is to compare and evaluate several compression techniques, including R3D3.

4.4 Related Work

In order to make easier the understanding of the driving forces and evolution of compact data structures, we followed a historical timeline in this chapter. As part of this journey, we already became familiar with the most fundamental concepts and related studies along the way. Nevertheless, we think that a selected set of sources deserve special attention as they smoothed the way of the author to build knowledge on the field of information theory and succinct data structures.

First of all, we refer the avid reader to the personal page and dissertation of Alex Bowe [120, 121] which gives a general introduction to succinct data structures and their applications. A more detailed survey of Navarro [86] outlines the basics and importance of wavelet trees and it also goes through "the surprising number of applications" where they are found to be useful. Furthermore, we shall not forget the paper [93] that introduced the improved version of RRR that became the main objective of R3D3 to compete with.

Finally, we suggest the reader to study and play with the freely available Succinct Data Structure Library (SDSL) [95] by Simon Gog that offers quick and easy onboarding to the practical experiments. Besides, studying the source code delivers incomparable experience to the user when the goal is to gain deep understanding of the available structures and to discover efficient optimization techniques.

Chapter 5

Summary

5.1 Conclusion

Network reliability and fast data processing have always been in the main focus of the telecommunication industry. This work presents both theoretical and experimental results to contribute to the existing literature. In the first part of the Dissertation, we have pointed out that the only commercially available IPFRR scheme, LFA, suffers from the drawback that the level of protection inherently depends on the layout of the given topology. To overcome this limitation, we presented two strategies for network augmentation in Chapter 2-3.

First, we introduced the *General LFA Graph Extension Problem* that asks for the minimum set of complement edges, that if added to the network under the model of correlated link-failures, improves LFA coverage the most, or even brings it to 100%. We determined the solvability of the problem and characterized its complexity. By extending the bipartite graph model to the case of multiple failures, we could apply several heuristics to obtain approximate solutions. We found that MSBT selects the smallest number of complement edges for attaining 100% coverage, while on the other hand, if the goal is to raise coverage as much as possible only with a few additional links, LJC clearly performs the best. For both link and node-protecting cases we found that the presence of SRGs poses minor (at most $7-12$ links) overhead comparing to the no-SRG setups. Finally, we verified that in the majority of the cases the problem could be solved to optimality (or very close to it) that makes LFA Graph Extension an appealing option to operators willing to make minor adjustments to their network.

Later, we investigated the potential of router virtualization that offers a flexible solution for operators having strict concerns about topological changes or additional hardware investments. We introduced RIOD, the *Resilient IP Overlay Design* problem and showed that it can be solved for all the inputs. We proposed a novel heuristic framework that is able to find a solution that can be tuned between optimal performance or fast execution time. Our numerical evaluation revealed that the majority of networks gain perfect LFA protection just by provisioning the same size of overlay as the original network (i.e. each router need to run only a single virtual instance in average). By restricting the number of virtual instances, we found minimal performance gap $(5 - 7\%)$ between the ILP and heuristic and we discovered that only a couple of new virtual nodes are sufficient to boost LFA coverage to $\sim 95\%$. Finally, we verified that the proposed framework indeed keeps the length of the virtual detours small whereupon the additional capacity overhead that a device suffers due to a failure is minimal.

The last goal of the Dissertation was to improve the performance of data queries for applications working on massive volumes of data. In Chapter 4, we discussed the essential challenge of data compression, called the space-time trade-off, then we reviewed the existing techniques that aim to give different answers to the same problem. We found that these techniques mainly focus on the compressibility of the data and leave the question open about how to achieve space reduction on the index. To close this gap, we presented R3D3 that combines the Elias-Fano coding scheme with the RRR indexing method to allow the usage of larger block sizes compared to conventional RRR implementations. In our measurements, we experienced notable performance gain at the price of a slight increase in the overall storage size as opposed to the best-known RRR representation schemes. Finally, we concluded that our results could encourage further study or optimization strategies to discover.

5.2 Theses Summary

In this section, we summarize the main results of the Dissertation and we present the theses. Our first goal was to increase the robustness of IP networks by installing complement edges that raise the overall number of LFAs.

Thesis 1. *I have formally defined the correlated failure tolerant LFA graph extension* *problem,* minLFA$_{\mathrm{SRG}}$*. I have proposed pre-processing and approximation algorithms with provable worst-case error bound, and I have provided extensive empirical evidence suggesting that the proposed algorithms are able to attain 100% LFA protection by adding average 30% of the links already existing in the network in the link-protecting case and average 60% in the node-protecting case. I have shown that there is a dependency between the number of necessary complement edges and the density of Shared Risk Groups.*

Discussed in Chapter 2.

Thesis 1.1. *[C3] The* minLFA$_{\mathrm{SRG}}$ *problem is NP-complete.*

See Theorem 1 in Section 2.3.2.

Thesis 1.2. *[J3, C3] I have shown a polynomial time pre-processing algorithm for* minLFA$_{\mathrm{SRG}}$ *that terminates with optimal result.*

The claim is available in Section 2.3.3. The problem is depicted on Fig. 2.7 and the algorithm is demonstrated in Fig 2.8.

Thesis 1.3. *[J3, C3] I have defined an $O(n^3)$ time construction that creates a proper bipartite graph representation of* minLFA$_{\mathrm{SRG}}$ *in the node-protecting case.*

See Section 2.3.4 and Theorem 3.

Thesis 1.4. *[J3, C2, C3] By extensive simulation studies, I have shown that the heuristics proposed from the literature are capable to attain perfect link-protecting LFA coverage in small and middle size networks with adding only a 20–40% of the number of links, while to achieve perfect node-protecting LFA coverage 50–70% of new links are needed, and in most of the cases the overshoot to the optimum is at most 5–15%. I have found that the LJC and MSBT heuristics are the most efficient amongst the proposed approximating algorithms.*

Discussed in Section 2.3.6.

The second main result also addresses the problem of resiliency in pure IP networks, and our suggestion is to provision virtual nodes in order to increase the number of available LFAs in the network.

Thesis 2. *I have defined the Resilient IP Overlay Problem, called* RIOD*, which aims to provision a virtual overlay on top of IP networks in order to achieve perfect LFA protection. I have shown that the complexity of the problem is NP-complete, and it is always solvable for any given input. I have introduced an algorithmic framework that can be parameterized either to return minimal number of virtual elements or to terminate in polynomial time. By performing numerical simulations on a set of input networks, I have found that on average a single virtual per physical router is enough to reach full protection.*

Discussed in Chapter 3.

Thesis 2.1. *[J2, C1] For a given* 2*-connected graph* G_S *with positive costs* c *there always exists an overlay* G_V *and cost setting* c_V *that solves* $\mathrm{RIOD}(G_S, c, V_S, \infty)$ *with* $\eta(G_V, c_V) = 1$.

The claim and the proof can be found in Section 3.3.3.

Thesis 2.2. *[J2, C1] The* $RIOD(G_S, c, U, k, \eta_{min})$ *problem is NP-complete.*

See Theorem 5 in Section 3.3.3.

Thesis 2.3. *[J2, C1] I have defined an algorithmic framework that, for any given number of virtual routers* k*, makes it possible to solve* $\mathrm{RIOD}(G, c, U, k)$ *with trading away the running time of the algorithm for the LFA-coverage attained by adding at most* k *virtual routers to the network. I have shown a construction that terminates at most* $O(n^5)$ *steps.*

We present the algorithm in Section 3.3.4. In Theorem 6 we suggest to add multiple virtual instances in every iteration to the network, and in the rest of the Section we optimize the running time of the algorithm. See Theorem 7.

Thesis 2.4. *[J2, C1] I have presented empirical evaluation to show that the proposed ILP adds only around* 30% *of the physical nodes to reach the* 95% *level of LFA protection. At the same time, to reach perfect protection, provisioning* 70% *virtual nodes are enough. On the other hand, when the goal is to minimize running time, I have found that the heuristic reaches the same* 95% *coverage by adding* 36% *of the physical nodes on average. Similarly, the heuristic requires* 85% *of the physical nodes to gain perfect protection that is* 15% *approximation error. I have shown that there is only around 30% difference in the length of recovery vs. default shortest paths.*

The numerical evaluation is available in Section 3.3.5.

The final set of results improves the execution time of selected operations on compressed data structures. The results suggest that combining the indexing structure of RRR with the the Elias–Fano encoding scheme brings performance benefits to computing devices.

Thesis 3. *I have proposed a novel succinct data structure that combines the storage scheme of RRR and the Elias-Fano encoding scheme. I have verified with analytical methods that the structure attains entropy-constrained size both on the data and the index, and I have given the complexity of* `access`*,* `rank` *and* `select` *queries. My numerical evaluation suggests that the data structure enables at most* 25% *faster operations at the price of a slight increase (around* 15%*) in the storage size.*

Discussed in Chapter 4.

Thesis 3.1. *[J1] I have shown that R3D3 encodes a t bitvector of length n of arbitrarily chosen block size, b, in*

$$nH_0 + np + \frac{n}{b}\left(2 + \log b\right) \quad \textit{bits.} \tag{5.1}$$

.

The claim, Theorem 8, and the proof are available in Section 4.2.3.

Thesis 3.2. *[J1] I have given a block size setting for R3D3, $pb = O(\log n)$, that achieves compression on the RRR indexing scheme by encoding t in*

$$nH_0 + nH_0\left(\frac{1}{2} + O\left(\frac{\log\log n}{\log n}\right)\right) \tag{5.2}$$

bits and supports `access`*,* `rank`*, and* `select` *operations in expected $O(\log n)$ time.*

See Theorem 9.

Thesis 3.3. *[J1] I have performed extensive numerical evaluations on a wide range of synthetic and real data to compare the performance characteristics of R3D3 to the best-known succinct compression schemes. I have found significant performance gap* *in the block decoding components of RRR and R3D3 in favor of R3D3. Measurements on synthetic data revealed* $2-3\%$ *space reduction compared to RRR that brings no significant gain for* **`rank`** *but improves 3 times the speed of* **`select`**. *Complex data structures having higher densities makes R3D3 perform* 25% *better, but only at the price of a* $10-15\%$ *(up to* 40%*) increase on the storage size.*

We discuss the results in Section 4.2.4.

Publications

Journal Papers

[J1] **M. Nagy**, J. Tapolcai and G. Rétvári. “R3D3: A Doubly Opportunistic Data Structure for Compressing and Indexing Massive Data”. *Infocommunications Journal*, pp. 58-66., 2019. (4/2 = 2)

[J2] **M. Nagy**, J. Tapolcai and G. Rétvári. “Node Virtualization for IP Level Resilience”. *IEEE/ACM Transaction on Networking*, 2018. (6/2 = 3)

[J3] **M. Nagy**, J. Tapolcai and G. Rétvári. “Optimization Methods for Improving IP-level Fast Protection for Local Shared Risk Groups with Loop-Free Alternates”. *Telecommunication Systems 56*, pp. 103-119., 2014. (6/2 = 3)

[J4] L. Csikor, **M. Nagy** and G. Rétvári. “Network Optimization Techniques for Improving Fast IP-level Resilience with Loop-Free Alternates”. *Infocommunications Journal*, pp. 2-10., 2012. (4/2 = 2)

Conference Papers

[C1] **M. Nagy**, J. Tapolcai and G. Rétvári. “On the Design of Resilient IP Overlays”. In *Proc., DRCN 2014*, Ghent, Belgium, 1-3. April 2014. (3/2 = 1.5)

[C2] **M. Nagy** and G. Rétvári. “IP hálózatok védelmének optimalitása többszörös hibák esetére”. In *Proc., Mesterpróba*, Budapest, Hungary, May 2012. (1/1 = 1)

[C3] **M. Nagy** and G. Rétvári. “An evaluation of approximate network optimization methods for improving IP-level fast protection with Loop-Free Alternates”. In *Proc., RNDM 2011*, Budapest, Hungary, September 2011. (3/1 = 3)

Other Publications

[C4] M. Szalay and **M. Nagy** and D. Géhberger and Z. Kiss and P. Mátray and F. Németh and G. Pongrácz and G. Rétvári and L. Toka. “Industrial-scale Stateless Network Functions”. In *Proc., IEEE Cloud*, Milan, Italy, 2019. (3/8 = 0.37)

Patents

[P1] **M. Nagy** and D. Fiedler and D. Géhberger and P. Mátray and G. Németh and B. Pinczel “Fast session restoration for latency sensitive middleboxes”. International Application No. PCT/IB2019/060031, TELEFONAKTIEBOLAGET LM ERICSSON, 2018. (2/6 = 0.33)

[P2] **M. Nagy** and D. Fiedler and D. Géhberger and P. Mátray and G. Németh and B. Pinczel and A. Császár “N+1 redundancy for virtualized services with low latency failover”. International Application No. PCT/IB2019/060037 , TELEFONAKTIEBOLAGET LM ERICSSON, 2018. (2/7 = 0.28)

Total publication score: 16.48

Glossary

E Set of edges in G. 16

E_S The set of physical edges in case of virtualization.

E_V The set of physical and virtual edges in case of virtualization.

G Simple, weighted, symmetric directed graph. 16

G_S The original physical graph in case of virtualization. 42

G_V The original physical graph extended with virtual elements in case of virtualization. 42

G_p Subgraph of G that needs pre-processing for being able to solve minLFA$_{\mathrm{SRG}}$. 27

H Entropy (information theory). 62

U_l A set of connected nodes in G_S with size l.

V Set of nodes in G. 16

V_S The set of physical nodes in case of virtualization. 42

V_V The set of physical and virtual nodes in case of virtualization. 42

GLFAVirt The General LFA Virtual Router Augmentation problem, where in each step we create multiple virtual instances. 49

RIOD The Resilient IP Overlay Design problem. 45

η_{LP} Link-protecting LFA coverage (i.e. ratio of link-protected vs. all node pairs). 20

η_{NP} Node-protecting LFA coverage (i.e. ratio of node-protected vs. all node pairs). 20

minLFA$_{\text{SRG}}$ The correlated failure tolerant LFA graph extension problem. 22

`popcount` Number of set bits in a given bitvector.

minSC Minimum set cover problem on bipartite graphs. 24

RRR A succinct data structure named after its inventors, Raman, Raman and Rao. 66

SD-array A succinct data structure treating *sparse* and *dense* sets separately. 68

Bibliography

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2015–2020," White Paper, Jun 1, 2016.

[2] L. Humphreys, T. von Pape, and V. Karnowski, "Evolving Mobile Media: Uses and Conceptualizations of the Mobile Internet," *Journal of Computer-Mediated Communication*, 2013.

[3] R. Felder, "All-ip, all-mobile business communication," Patton-Inalp Networks, Tech. Rep., June 2015.

[4] J. Moy, "Ospf version 2," RFC 2328, Apr 1998.

[5] D. Oran, "Osi is-is intra-domain routing protocol," RFC 1142, Febr 1990.

[6] G. Iannaccone, C.-N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an ip backbone," in *ACM SIGCOMM Internet Measurement Workshop*, 2002, pp. 237–242.

[7] M. Shand and S. Bryant, "IP Fast Reroute framework," RFC 5714, Jan 2010.

[8] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-Free Alternates," RFC 5286, 2008.

[9] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, ser. Series in Computer Science. Addison Wesley, 1998.

[10] Erik Demaine, "Advanced Data Structures," 2012, massachusetts Institute of Technology: MIT OpenCourseWare.

[11] T. Socolofsky and C. Kale, "A tcp/ip tutorial," RFC 1180, Jan 1991.

[12] H. Balakrishnan, “Wide-area internet routing,” Course Text, MIT, 6.033, January, 2009.

[13] “Software-Defined Networking,” https://www.opennetworking.org/sdn-definition/.

[14] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[15] M. Amarif and I. Alashoury, “The implicit path cost optimization in dijkstra algorithm using hash map data structure,” in *International conference on Computer Sciences & Infromatioon Technology COSIT2019*, 2019, pp. 33–44.

[16] B. Fortz, J. Rexford, and M. Thorup, “Traffic engineering with traditional IP routing protocols,” *IEEE Comm. Mag.*, vol. 40, no. 10, pp. 118–124, Oct 2002.

[17] G. Swallow and S. Bryant, “Avoiding Equal Cost Multipath Treatment in MPLS networks,” RFC 4928, Jun 2007.

[18] M. Shand and S. Bryant, “A framework for loop-free convergence,” RFC 5715, Jan 2010.

[19] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol label switching architecture,” RFC 3031, 2001.

[20] I. Minei, , and J. Lucek, *MPLS-Enabled Applications: Emerging Developments and New Technologies, 3rd Edition.* John Wiley & Sons, Ltd., 2010.

[21] I. Minei, “Scaling considerations in mpls networks,” North American Network Operators Group (NANOG), Juniper Networks, Tech. Rep., October 2005.

[22] P. Lapukhov, “Scaling in MPLS Networks,” 2010, available online: http://blog.ine.com/2010/08/16/scaling-mpls-networks/.

[23] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah, “Proactive vs reactive approaches to failure resilient routing,” in *INFOCOM*, 2004.

[24] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C.-N. Chuah, “Failure inferencing based fast rerouting for handling transient link and node failures,” in *INFOCOM*, 2005.

[25] G. Enyedi, G. Rétvári, and T. Cinkler, "A novel loop-free IP fast reroute algorithm," in *EUNICE*, 2007.

[26] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using Not-via addresses," Internet Draft, March 2010.

[27] A. Li, P. Francois, and X. Yang, "On improving the efficiency and manageability of NotVia," in *ACM CoNEXT*, 2007.

[28] G. Enyedi, P. Szilágyi, G. Rétvári, and A. Császár, "IP Fast ReRoute: lightweight Not-Via without additional addresses," in *INFOCOM Mini-conf*, 2009.

[29] A. Atlas, R. Kebler, M. Konstantynowicz, G. Enyedi, A. Császár, and M. Shand, "An architecture for IP/LDP fast-reroute using maximally redundant trees," Internet Draft, Oct 2011.

[30] T. Čičic, A. F. Hansen, and O. K. Apeland, "Redundant trees for fast IP recovery," in *Broadnets*, 2007, pp. 152–159.

[31] G. Enyedi and R. Rétvári, "Finding multiple maximally redundant trees in linear time," *to appear in Periodica Polytechnica, available online: http://opti.tmit.bme.hu/~enyedi/ipfrr/distMaxRedTree.pdf*, 2010.

[32] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne, "Multiple routing configurations for fast IP network recovery," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 473–486, 2009.

[33] G. Schollmeier, J. Charzinski, A. Kirstadter, C. Reichert, K. Schrodi, Y. Glickman, and C. Winkler, "Improving the resilience in IP networks," in *High Performance Switching and Routing, 2003, HPSR. Workshop on*, 2003, pp. 91–96.

[34] K.-W. Kwong, L. Gao, R. Guerin, and Z.-L. Zhang, "On the feasibility and efficacy of protection routing in IP networks," in *INFOCOM 2010, long version appears as Tech. Rep. 2009, University of Pennsylvania*, 2010.

[35] A. Császár, G. Enyedi, M. Hidell, G. Rétvári, and P. Sjödin, "Converging the evolution of router architectures and IP networks," *IEEE Network Magazine,*

Special Issue on Advances in Network Systems Architecture, vol. 21, no. 4, pp. 8–14, July-August 2007.

[36] Cisco Systems, "Cisco IOS XR Routing Configuration Guide for the Cisco CRS Router, Release 4.2," 2012.

[37] Hewlett-Packard, "HP 6600 Router Series: QuickSpecs," 2008, available online: http://h18000.www1.hp.com/products/quickspecs/13811_na/13811_na.PDF.

[38] Juniper Networks, "JUNOS 12.3 Routing protocols configuration guide," 2012.

[39] A. Markopoulou, G. Iannacone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *Proc. IEEE Infocom*, Mar. 2004.

[40] D. Watson, F. Jahanian, and C. Labovitz, "Experiences with monitoring OSPF on a regional service provider network," in *ICDCS*, 2003, pp. 204–212.

[41] G. Iannaccone, C. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of IP restoration in a tier-1 backbone," *IEEE Network*, vol. 18, no. 2, pp. 13–19, 2004.

[42] T. Gomes, C. Simoes, and L. Fernandes, "Resilient routing in optical networks using SRLG-disjoint path pairs of min-sum cost," *Springer Telecommunication Systems Journal*, 2010.

[43] M. Nagy, "Github homepage," https://nmate.github.io, 2019.

[44] "LEMON – Library for Efficient Modeling and Optimization in Networks," http://lemon.cs.elte.hu/, 2014.

[45] "GUROBI – Linear Programming Solver," http://www.gurobi.com, 2018.

[46] "BOOST – C++ Libraries," http://www.boost.org, 2018.

[47] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *ACM IMC*, 2002, pp. 231–236.

[48] SNDlib, "Survivable fixed telecommunication network design library," http://sndlib.zib.de.

[49] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," http://www.topology-zoo.org.

[50] M. Garey, , and D. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1990.

[51] A. P. Punnen, *The Traveling Salesman Problem: Applications, Formulations and Variations.* Boston, MA, USA: Springer, 2007, pp. 1–28.

[52] Erik Demaine, Srini Devadas, and Nancy Lynch, "Design and Analysis of Algorithms," 2015, massachusetts Institute of Technology: MIT OpenCourseWare.

[53] G. Rétvári, J. Tapolcai, G. Enyedi, and A. Császár, "IP Fast ReRoute: Loop Free Alternates revisited," in *INFOCOM 2011*, 2011, pp. 2948–2956.

[54] B. Mazbic-Kulma and K. Sep, "Some approximation algorithms for minimum vertex cover in a hypergraph," in *Computer Recognition Systems 2*, ser. Advances in Soft Computing, M. Kurzynski, E. Puchala, M. Wozniak, and A. Zolnierek, Eds. Springer Berlin / Heidelberg, 2007, vol. 45, pp. 250–257.

[55] L. Lovász, "On the ratio of optimal integral and fractional covers," *Discrete Mathematics*, vol. 13, no. 4, pp. 383–390, 1975.

[56] P. Kulaga, P. Sapiecha, and K. Sej, "Approximation Algorithm for the Argument Reduction Problem," in *Computer recognition systems: proceedings of the 4th International Conference on Computer Recognition Systems, CORES'05.* Springer Verlag, 2005, p. 243.

[57] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, "Fast recovery mechanisms in the data plane," in *IEEE Communications Surveys and Tutorials*, 2020.

[58] "The future internet assembly (fia)," https://link.springer.com/book/10.1007/978-3-642-38082-2#about, 2013.

[59] L. Csikor, G. Rétvári, and J. Tapolcai, "High availability in the Future Internet," in *The Future Internet*, ser. Lecture Notes in Computer Science, A. Galis and A. Gavras, Eds. Springer Berlin Heidelberg, 2013, vol. 7858, pp. 64–76.

[60] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So, "Remote loop-free alternate (lfa) fast reroute (frr)," Internet Requests for Comments, IETF, RFC 7490, April 2015.

[61] G. Rétvári, L. Csikor, J. Tapolcai, G. Enyedi, and A. Császár, "Optimizing IGP link costs for improving IP-level resilience (best paper award)," in *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, October 2011, paper http://lendulet.tmit.bme.hu/~retvari/publications/drcn_2011.pdf, pp. 62–69.

[62] Cisco, "Cisco catalyst 4500 series switch software configuration guide, 15.0," 2016.

[63] O. Narmanlioglu and E. Zeydan, "Software-defined networking based network virtualization for mobile operators," *Computers & Electrical Engineering*, vol. 57, pp. 134 – 146, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790616302543

[64] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, p. 862–876, Apr. 2010. [Online]. Available: https://doi.org/10.1016/j.comnet.2009.10.017

[65] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central Control Over Distributed Routing," in *ACM SIGCOMM*, London, UK, August 2015.

[66] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *Communications Magazine, IEEE*, vol. 53, pp. 90–97, 02 2015.

[67] M. Nagy, J. Tapolcai, and G. Rétvári, "Node virtualization for IP level resilience," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2018.

[68] J. Tapolcai and G. Rétvári, "Router Virtualization for Improving IP-level Resilience," in *INFOCOM 2013*, 2013.

[69] R. Hartley, "Transmission of Information," in *International Congress of Telegraphy and Telephony, Italy*, 1927.

[70] C. E. Shannon, “A mathematical theory of communication,” *Bell Systems Technical Journal*, vol. 27, pp. 623–656, 1948.

[71] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, “Big data: The next frontier for innovation, competition, and productivity,” McKinsey Global Institute, Tech. Rep., June 2011.

[72] O. Trelles, P. Prins, M. Snir, and R. C. Jansen, “Big data, but are we ready?” *Nat Rev Genet*, vol. 12, no. 3, p. 224, 2009.

[73] G. Jacobson, “Space-efficient static trees and graphs,” in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, ser. FOCS ’89, 1989, pp. 549–554.

[74] G. Navarro and V. Mäkinen, “Compressed full-text indexes,” *ACM Comput. Surv.*, vol. 39, no. 1, 2007.

[75] P. B. Miltersen, “Lower bounds on the size of selection and rank indexes,” in *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA ’05, 2005, pp. 11–12.

[76] A. Golynski, *Optimal lower bounds for rank and select indexes.* Elsevier, 2007.

[77] D. Clark, *Compact Pat Trees*, ser. PhD thesis. University of Waterloo, Canada, 1996.

[78] R. Grossi, A. Orlandi, R. Raman, and S. Rao, “More haste, less waste: Lowering the redundancy in fully indexable dictionaries,” in *26th International Symposium on Theoretical Aspects of Computer Science STACS*, 2009, pp. 517–528.

[79] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS ’00, 2000.

[80] A. C. Yao, “Should tables be sorted?” *Journal of Association for Computing Machinery*, vol. 28, no. 3, pp. 615–628, July 1981.

[81] V. R. R. Raman and S. R. Satti, "Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets." *ACM Transactions on Algorithms*, vol. 3(4), 2007.

[82] A. Brodnik and I. Munro, "Membership in constant time and almost-minimum space," *Journal on Computing*, vol. 28, no. 5, pp. 1627–1640, 1999.

[83] R. Pagh, "Low redundancy in static dictionaries with constant query time," *Journal on Computing*, vol. 31, no. 2, pp. 353–363, 2001.

[84] M. Pătraşcu, "Succincter," in *49th FOCS*, 2008, pp. 305–313.

[85] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '03, 2003, pp. 841–850.

[86] G. Navarro, "Wavelet trees for all," in *Journal of Discrete Algorithms*, ser. Volume 25, 2014, pp. 2–20.

[87] V. Mäkinen and G. Navarro, "New search algorithms and time/space tradeoffs for succinct suffix arrays," University of Helsinki, Tech. Rep., 2004.

[88] P. Elias, "Efficient storage and retrieval by content and address of static files," *J. Assoc. Comput. Mach.*, vol. 21, pp. 246–260, 1974.

[89] S. Vigna, "Quasi-succinct indices," in *ACM International Conference on Web Search and Data Mining, WSDM*, 2013, pp. 83–92.

[90] D. Okanohara and K. Sadakane, "Practical entropy-compressed rank/select dictionary," in *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, 2007, pp. 60–70.

[91] S. Gog, "Compact and succinct data structures: From theory to practice," available online: http://es.csiro.au/ir-and-friends/20131111/anu_gog_seminar.pdf, 2015.

[92] F. Claude and G. Navarro, "Practical rank/select queries over arbitrary sequences," in *Proc. 15th International Symposium on String Processing and Information Retrieval (SPIRE)*, ser. LNCS 5280. Springer, 2008, pp. 176–187.

[93] G. Navarro and E. Providel, *Fast, Small, Simple Rank/Select on Bitmaps*. Springer Berlin Heidelberg, 2012, pp. 295–306.

[94] D. E. Knuth, *The Art of Computer Programming: Combinatorial Algorithms*, ser. Series in Computer Science. Addison-Wesley, 2011.

[95] S. Gog, "Succinct Data Structure Library," https://github.com/simongog/sdsl-lite.

[96] A. Majdán and G. Rétvári, "Development and performance evaluation of fast combinatorial unranking implementations," in *Meeting of the European Network of Universities and Companies in Information and Communication Engineering, EUNICE*, 2014.

[97] D. Meyer, L. Zhang, and K. Fall, "Report from the iab workshop on routing and addressing," RFC 4984, 2007.

[98] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger, "Compressing IP forwarding tables: towards entropy bounds and beyond," in *ACM SIGCOMM*, 2013, paper http://lendulet.tmit.bme.hu/~retvari/publications/sigcomm_2013_tech_rep.pdf, pp. 111–122.

[99] M. Nagy, "R3D3-Fast Compressed Bitvectors for the Succinct Data Structure Library," https://github.com/nmate/sdsl-lite.

[100] P. Ferragina, G. Navarro, "Pizza & Chili Corpus," http://pizzachili.dcc.uchile.cl/.

[101] UCSC Genome Bioinformatics, "The UCSC Genome Browser," http://hgdownload.cse.ucsc.edu/downloads.html.

[102] I. Witten, T. Bell, and J. Cleary, "The Calgary Corpus," 1987, http://corpus.canterbury.ac.nz/descriptions/#calgary.

[103] Microsoft, "SQL Server 2017," 2017, available online: https://docs.microsoft.com/en-us/sql/relational-databases/search/create-and-manage-full-text-indexes.

[104] MySQL Server, "InnoDB Fulltext Indexes," 2017, available online: https://dev.mysql.com/doc/refman/5.6/en/innodb-fulltext-index.html.

[105] Maria DB, "Full-Text Index Overview," 2018, available online: https://mariadb.com/kb/en/library/full-text-index-overview.

[106] S. Büttcher, C. Clark, and G. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines.* MIT Press, 2010.

[107] A. D., G. S., and O. M., "Compressed Self-indices Supporting Conjunctive Queries on Document Collections," in *String Processing and Information Retrieval, SPIRE'10*, vol. 6393. Springer, 2010.

[108] G. Navarro and Y. Nekrich, "Top-k document retrieval in optimal time and linear space," in *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012, pp. 1066–1078.

[109] F. Claude and G. Navarro, "Extended compact web graph representations," in *Algorithms and Applications (Ukkonen Festschrift)*, ser. LNCS 6060, T. Elomaa, H. Mannila, and P. Orponen, Eds. Springer, 2010, pp. 77–91.

[110] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel, "Iterative dictionary construction for compression of large DNA data sets," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 137–149, Jan 2012.

[111] M. C. Brandon, D. C. Wallace, and P. Baldi, "Data structures and compression algorithms for genomic sequence data," *Bioinformatics*, vol. 25, no. 14, pp. 1731–1738, 2009.

[112] V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki, "Storage and retrieval of individual genomes," in *Proc. 13th Annual International Conference on Computational Molecular Biology (RECOMB)*, ser. LNCS 5541, 2009, pp. 121–137.

[113] R. Raman, "Succinct data structures for data mining," Workshop on Algorithms for Large-Scale Information Processing in Knowledge Discovery, 2014.

[114] F. Claude and G. Navarro, “Compressed Data Structure Library,” https://github.com/fclaude/libcds2.

[115] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” *CoRR*, vol. abs/1712.01208, 2017. [Online]. Available: http://arxiv.org/abs/1712.01208

[116] R. Agarwal, A. Khandelwal, and I. Stoica, “Succinct: Enabling queries on compressed data,” in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’15, 2015, pp. 337–350.

[117] MongoDB, 2018, available online: https://www.mongodb.com.

[118] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” *Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[119] G. Rétvári and A. Kör osi and J. Tapolcai, “The Internet Routing Entry Monitor,” http://lendulet.tmit.bme.hu/fib_comp/.

[120] A. Bowe, *Multiary Wavelet Trees in Practice*, ser. PhD thesis. RMIT Univeristy, Australia, 2010.

[121] A. Bowe, “Alex bowe — stuff i’m thinking about,” https://alexbowe.com.

List of Tables

List of Figures