Summary

In a growing digital age, more and more data is being sent through modern data networks, approaching the limits of traditional networks. Research has therefore focused on ways to decrease the bottle-necking of traditional routing technology, instead opting for the use of light in optical fibers to transmit data at very high bandwidth. However, while bandwidth is important, creating a failure resilient network is just as critical in order to uphold quality-of-service standards and reliability. While the field of failure resilience is a mature topic for traditional networks, these methods are not one-to-one applicable in optical networks. Furthermore, existing research for failure resilience in optical networks relies on over-allocation of resources, which hinders the overall capacity of the network. As such, ensuring failure resilience with efficient resource allocation still remains a challenge in optical networks.

In this paper, we set out to explore how to heighten the network resilience of elastic optical networks. An elastic optical network consists of routers connected by links, where each link has a capacity for data transmission represented by a number of spectrum slots. In the network, data is represented by traffic demands that each has an ingress and egress router in addition to an amount of data being transported through that demand. Given an optical network and a set of traffic demands, the task is to find a path (sequence of links) and spectrum allocation (sequence of slots) for each demand such that the highest used slot in the entire network is minimized. To find valid solutions to the RSA problem, three constraints are enforced: (1) for a given demand, the spectrum allocation must be continuous along the edges of the assigned path; (2) slots allocated to a demand must be consecutive, and (3) any two demands cannot occupy the same slot on any link in the network.

We encode the RSA problem as a Boolean function with Boolean variable encodings of the possible path assignments and slot allocations. The idea is that the assignments to these variables represent valid solutions to the given RSA problem if that assignment makes the Boolean function evaluate to *true*. We use the data structure Binary Decision Diagrams (BDDs) to efficiently compute and store all valid assignments to a given Boolean function. Thereby, in contrast to previous work on solving the RSA problem, we are able to compute and compactly represent all solutions to a given RSA problem.

It is, however, not required to have all solutions represented for the purpose of failure resilience, and we can thus speed up the build time for our BDDs by limiting the number of solutions that must be represented. In particular, the baseline BDD contains redundant channel assignments that can be pruned without losing the required solutions. Our results show that that these improvements increase the number of demands that can be solved. Compared to ILP models, we are able to solve for fewer demands, but this is expected, as these approaches only find singular solutions.

From solving the RSA problem, the result is a BDD that has at least one optimal solution for each possible path assignment and it can thus be used when optical networks experience failures, meaning some links become unavailable during transmission. Handling these failures fast is crucial and creates a necessity for link-failure resilience. Link-failure resilience entails being able to find a new solution (path and slot allocation) if one exists for a given failure scenario. In turn, k-link-failure resilience entails being link-failure resilient for up to k failed links, and perfect resilience entails the former for an arbitrary number k. Our work stands in contrast to previous work, which focuses on single link failures or k-link failures for non arbitrary values of k.

Querying the BDD for a new solution in case of link-failures consists of two steps. (1) finding the subtree that contains the solutions which do not use any of the failed links, (2) is finding an optimal solution in this subtree. We implement 2 ways of the first querying step: (1) We prune invalid solutions by disallowing all paths using failed links. (2) We use variables to directly encode in the BDD possible edges that can fail. Querying then works by only querying on those added auxiliary

variables. Our results, show that method (2) incurs a longer build time, but drastically improves the query time.

We also show how an intermediate step can be added to the querying process, where the subtree can be further pruned to only contain solutions that comply with a specific property. In our case, the property enforced is that the demands which are unaffected by the specific link failures changes neither their path or spectrum allocation. This intermediate pruning, however, induces a longer query time.

We implement our BDD based approaches to the RSA and k-link failover problem in a prototype implementation of the tool ExpectAll, and compare it with an state-of-the-art ILP model on two metrics. Firstly, the time it takes to find one solution in link failure scenarios, where it is found that the ILP model is too slow for practical use, while both our query approaches are capable of finding a new solution within well-established expected recovery times. Secondly, we compare the times it takes to build the BDDs with the time to precompute the required solution for all k-link failure scenarios. We find that the ILP approach is feasible for small k, but having to precompute for an arbitrary high number of link failures takes too long due to the exponentially many combinations of link failures the ILP have to find a solution for. Our approach of computing the BDD representing many solutions at once and querying when a link failure occurs, is thus able to outperform the ILP model for a small number of critical demands.

ExpectAll: A BDD Based Approach to Failure Resilience in Elastic Optical Networks

Gustav S. Bruhns, Martin P. Hansen, Rasmus Hebsgaard, and Frederik M. W. Hyldgaard

Aalborg University, Institute of Computer Science

Abstract. With the increasing demand for higher bandwidth and quality of service in modern networks, how to achieve fast, resilient networks is an important field of research. Here, recent advances in elastic optical networks have enabled fine-grained resource allocation for traffic demands, which introduced the Routing and Spectrum Allocation (RSA) problem. State-of-the-art methods for finding optimal solutions to the RSA problem are too slow for real-time practical applications, such as ensuring failure resilience, and faster methods cannot guarantee optimal resource allocation. Moreover, current methods for ensuring failure resilience in classic networking cannot be directly adapted to optical networks, and current methods for optical networks generally rely on over allocation of the spectrum to ensure rapid recovery. To this end, we present the tool ExpectAll, a novel approach based on binary decision diagrams (BDDs) to ensure failure resilience for multi-link failures without resorting to spectrum overallocation. Our method efficiently computes solutions to the RSA problem, facilitating optimal failover solutions for any failure scenario involving up to k links. ExpectAll surpasses state-of-the-art methods in both the speed of finding a single optimal solution during a failure and the preparation time required to compute sufficient solutions to ensure resilience for arbitrary large k-link failures. Additionally, since ExpectAll can compute and represent all potential solutions, it is adaptable for network operators to find solutions that meet specific desired properties.

Keywords: binary decision diagrams, routing and spectrum allocation, RSA, elastic optical network, failover, link failure resilience

1 Introduction

With more than two-thirds of the global population having access to the internet and the increasing amount of data that follows from more and more devices being connected, modern data networks are put under pressure, heightening the need for increased bandwidth and network resilience [1]. Traditionally, optical networks use wavelength-division multiplexing (WDM) [2] in order to split the frequency spectrum into slots of 50 GHz. As the amount of traffic has increased and is only predicted to increase more in the future, a new flexible paradigm has been proposed using elastic *Flexgrid* technology to enable more fine-grained splitting of the bandwidth down to 6.25 GHz slots [3].

In elastic optical networks, data is transported along *lightpaths*, which are connections between two access points in the network using one or more of the spectrum slots. Given a set of traffic demands, the routing and spectrum allocation (RSA) problem involves finding a lightpath in the form of a route through the network and a set of spectrum slots for each demand. Assuming that a network does not utilize optical converters, a solution to the RSA problem must comply with the constraints of:

- *Continuity*: A lightpath must use the same spectrum slots throughout its entire flow through the network.
- *Contiguity*: The spectrum slots used on a lightpath must be consecutive.
- *Non-overlapping*: For each link in the network, a spectrum slot can be used by at most one lightpath.

With the increasing scale of networks, and the amount of data that can be transported through an elastic optical network, the consequences of link outages become more severe [4, 5]. Examples include loss of business revenue and disruption of safety-critical networks [6, 7, 8, 9]. It is therefore important that networks quickly recover from link failures to reduce the consequences by finding an alternative routing for the demands affected by the link failures [10]. Current approaches achieve quick recovery times for link failures by creating a backup path for each demand [11, 12] through over-allocation, thereby wasting network resources. Furthermore, this approach can only handle one-link failures, but multiple links are likely to fail [13]; hence, preparing for only one link failures can be inadequate. Ensuring resilience to multiple link failures with quick recovery times in elastic optical networks therefore presents a relevant challenge in the foreseeable future.

Our Contributions. We design and implement ExpectAll [14], a novel approach using Binary Decision Diagrams (BDDs) to ensure failure resilience in elastic optical networks. The tool efficiently finds and compactly stores *all* solutions to the RSA problem, which can be leveraged to quickly provide real-time solutions to multiple link failure scenarios. To this end, our contributions are as follows.

As our first contribution, we investigate the use of integer linear programming (ILP) for ensuring failure resilience in optical networks without over-allocation. We experiment on two real network topologies for multiple-link failures and show that ILP is impractical for ensuring failure resilience, both in terms of the time for synthesizing new network configurations as well as memory requirements.

As our second contribution, we leverage the technology of BDDs to solve the RSA problem for the purpose of failure resilience, culminating in the tool ExpectAll. We prove that our solution finds all solutions and present improvements to increasing its scalability on the number of traffic demands.

As our third contribution, we showcase two applications of the BDD solution for ensuring resilience for multiple-link failures. The first application can handle an arbitrary number of link failures, whereas the other application provides quicker recovery times at the cost of being able to handle at most k link failures for a fixed k. The two applications are compared to the ILP approach, where it is clear that both outperform the ILP when comparing how quickly they are able to recover from link failures, as well as how they scale on the number of link failures.

Related work. The Routing and Wavelength Assignment (RWA) problem is a well-studied [15, 16, 17], NP-complete problem [18] in optical networks, dating back to the 90's [17], where the optical networks used Wavelength Division Multiplexing (WDM) technology [2] to partition the spectrum into a fixed number of wavelengths. Later on, the notion of fine-grained spectrum allocation was introduced for elastic optical networks as an improvement on the WDM optical networks. As a result, the routing and spectrum allocation (RSA) problem comes as an extension to the traditional RWA problem for WDM networks, due to the introduction of fine-grained spectrum allocation. The RSA problem is relatively new, dating back to the early 2010's, and has been proven to be NP-hard [19]. In the first iterations of solutions to the problem, Integer Linear Programming (ILP) is applied to formalize the lightpath constraints with the goal of minimizing the maximum slot index used on the spectrum [19, 20, 21]. Later iterations of ILPs focus on making the ILP formulations more concise and efficient; examples are Zhang et al. [22], which improves upon the work done in [20], Velasco Esteban et al. [23] that introduce the notion of channels to handle the spectrum contiguity constraint outside of their ILP formulation, and Wang et al. [24] in which they contribute with a relaxed ILP problem to establish a lower bound. In contrast to our solution, the ILP implementations are effective at finding a single optimal solution to a given RSA problem for a non-trivial number of demands. However, we find all optimal solutions and exceed the ILPs in recovery time. While ILP formulations find optimal solutions, they are generally not applicable for time-critical purposes, such as reacting to link failures. For more scalable approaches to solve the RSA problem, heuristics have been proposed [19, 25, 26], as well as genetic algorithms [3, 27, 28] and reinforcement learning models [29, 30]. These approaches trade off optimality for computation speed, where the genetic

algorithms tend to be closer to achieving optimal solutions than the heuristics and reinforcement learning approaches at the cost of being computationally slower. In contrast to these approaches, we are able to find all optimal solutions for path and spectrum allocations, which can prove useful for handling changes in a network setting, such as quickly recovering from link failures.

While the study of link failure resilience in traditional networks is well-researched (see overview in [4, 5]), the approaches cannot be applied one-to-one in elastic optical networks due to the additional complexity introduced by the aforementioned constraints. Furthermore, the approaches that have been proposed for ensuring link failure resilience in optical networks generally only prepare for one-link failures by introducing backup paths [11, 12, 31, 32, 33]. For example, Castro et al. [11] propose an MILP formulation that maximizes the total bitrate recovered in case of a single-link failure scenario by allocating a backup path for each demand such that when a demand is affected by a link failure, it can quickly switch over to its backup path, and Singhal et al. [33] and Gao et al. [31] expand upon this idea with the notion of the more resource-friendly cross-sharing, where groups of demands with link-disjoint primary paths are allowed to share a backup path. These approaches using backup paths assume one link failures, but some research (see e.g. Athe and Singh [34] and Li et al. [35]) has been carried out to extend the failure resilience to two link and network-bound link failures. Common for the current approaches to ensuring failure resilience is that they require resources to always be allocated for both the primary and backup paths in the network to ensure quick failover, which entails that they do not provide optimal solutions. Our approach finds optimal solutions while being able to handle more than one link failures without having to resort to resource over-allocation.

Organization. The rest of the paper is organized as follows. First, we formally define the RSA problem and the problem of handling link failures in Section 2. Then, in Section 3, we present and evaluate how to use an ILP formulation to handle link failures. In Section 4, we encode the RSA problem in BDDs which we then use in Section 5 to handle link failures. Finally, in Section 6, we compare our BDD-based approach with the ILP approach, and conclude on our work in Section 7.

2 Problem Definition

A *network topology* is a tuple $G = (V, E, src, tgt)$ where V is a finite set of nodes, E is a finite set of edges and $src, tgt : E \to V$ denote the source and target of an edge, where $src(e) \neq tgt(e)$.

Let a path be a sequence of connected edges $\pi = e_1e_2e_3...e_n$ such that $tgt(e_i) = src(e_{i+1})$ for $1 \leq i < n$. Then, let $e \in \pi$ denote that edge e is part of path π, the mappings $first, last : \mathbf{Paths} \to E$ denote the first and last edge on the path, respectively, and $\pi \cap \pi'$ denote the set of edges that two paths share. A *simple path* is a path where $src(e') \neq src(e)$ and $tgt(e) \neq tgt(e')$ for all pairs of distinct edges $e, e' \in \pi$; let **Paths** be the set of all simple paths in the topology G.

We assume a finite set of demands D with source and target nodes represented by the mappings $ingress, egress : D \to V$ respectively, and $size : D \to \mathbb{N}$ representing the amount of data of a demand.

Furthermore, we assume a finite set of spectrum slots $F = \{1, 2, .., f_{max}\}$. Finally, a channel is a finite set $C \subseteq F$ of consecutive slots, and $\mathbb{C}$ is the set of all channels. Then, all possible channels for a demand is represented by the mapping $channels : D \to 2^{\mathbb{C}}$.

Definition 1 (Routing and Spectrum Allocation Problem).
Given as input

- *a network topology* $G = (V, E, src, tgt)$ *with simple paths* $\boldsymbol{Paths}$,
- *a set of demands* $D = \{d_1, d_2, ..., d_m\}$,
- *a mapping* $DPaths : D \to 2^{\boldsymbol{Paths}}$ *of available paths for each demand* $d \in D$, *where for every* $\pi \in DPaths(d)$ *it holds that* $src(first(\pi)) = ingress(d)$ *and* $tgt(last(\pi)) = egress(d)$,

- *a finite set of available slots* $F = \{1, 2, .., f_{max}\}$,
- *and a modulation mapping* $\Delta : \boldsymbol{Paths} \to \mathbb{N}$ *which returns the number of slots required per sent unit of data on a path,*

find a solution (P, ω) *to the problem, where*

- P *is a total function* $P : D \to \boldsymbol{Paths}$ *such that* $P(d) \in DPaths(d)$ *for every demand* $d \in D$*, and*
- ω *is a total function* $\omega : D \to 2^F$*, such that* $\omega(d) \in channels(d)$*, and* $|\omega(d)| = \Delta(P(d)) \cdot size(d)$

such that for all pairs of distinct demands $d, d' \in D$ *either* $P(d) \cap P(d') = \emptyset$ *or* $\omega(d) \cap \omega(d') = \emptyset$.

Different metrics have been used to define optimal solutions to the RSA problem, such as minimizing unserved bandwidth of demands [23], the number of frequency slots used [19], and the highest frequency slot index used by any demand [21]. Since we focus on a set of few but critical demands, that should be allocated in such a way that there is amble room for the remaining less critical demands, we choose the latter. For a given ω we can find the highest used slot as $usage(\omega) = \max_{d \in D} \max \omega(d)$. A *network optimal solution* is then a solution to the RSA problem (P, ω) where $usage(\omega) \leq usage(\omega')$ for any other solution to the RSA problem (P', ω').

Example 1. Figure 1a illustrates a simple example of the RSA problem with a small network topology consisting of six nodes and seven edges and a spectrum width of two. There are two demands d_1 and d_2, both having *size* 1. Demand d_1 has nodes v_1 and v_5 as its source and target respectively, whereas demand d_2 has v_2 and v_6 as its source and target respectively. Each demand has two possible paths. If demand d_1 uses path π_1, then it must be allocated two frequency slots due to modulation. The same holds true for demand d_2 if it uses path π_3. Hence, there are three different possible channels for both demands, as seen on Figure 1a. An example of an optimal solution to this RSA problem is RSA Solution 1 where demand d_1 is assigned path π_0 and channel $\omega(d_1) = \{1\}$ while demand d_2 is assigned path π_2 and channel $\omega(d_1) = \{1\}$.

Having defined the RSA problem, we now formally define the problem of handling up to k link failures for a given RSA problem.

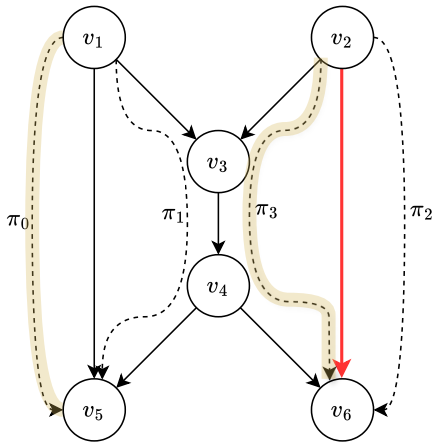
Definition 2 (RSA k-Link Failover Problem).
Given as input a set of link failures $E_{fail} \subseteq E$ *where* $|E_{fail}| \leq k$*, we want to find a solution* (P, ω) *to the RSA problem where* $P(d) \cap E_{fail} = \emptyset$ *for all* $d \in D$.

Example 2. An example of the k-link failover problem is shown on Figure 1a where the link between node v_2 and node v_6 has failed. The goal is to find a solution (P, ω) where no demand is assigned a path that uses the failed link. RSA solution 1 from Figure 1a is now not a valid solution, as d_2 is assigned path π_2 which uses the failed link. However, RSA solution 2 is a valid solution, as demand d_2 is assigned path π_3 which does not use the failed link, and demand d_1 is also assigned a path that does not use the failed link. This solution becomes an optimal solution in this failure scenario.

The goal is now to be capable of handling all possible k-link failure scenarios. In the next section, we present how one can do this using a state-of-the-art ILP formulation of the RSA problem.

3 Using ILP to Assure Failure Resilience

The main motivation behind this paper is to be able to provide an optimal solution in any failure scenario, avoiding over allocation in the network. In accordance with the Metro Ethernet Forum [10], we aim for an average recovery time of less than 50 ms with an upper limit of 200 ms when link



Problem:

$D = \{d_1, d_2\}$

$size(d_1) = size(d_2) = 1$

$DPaths(d_1) = \{\pi_0, \pi_1\}$

$DPaths(d_2) = \{\pi_2, \pi_3\}$

$$\Delta(\pi) = \begin{cases} 2 & \text{if } \pi \in \{\pi_1, \pi_3\} \\ 1 & otherwise \end{cases}$$

$F = \{1, 2\}$

$\mathbb{C} = \{\{1\}, \{2\}, \{1, 2\}\}$

$channels(d_1) = \{\{1\}, \{2\}, \{1, 2\}\}$

$channels(d_2) = \{\{1\}, \{2\}, \{1, 2\}\}$

Solution 1 (optimal):

$P(d_1) = \pi_0$

$P(d_2) = \pi_2$

$\omega(d_1) = \{1\}$

$\omega(d_2) = \{1\}$

Solution 2:

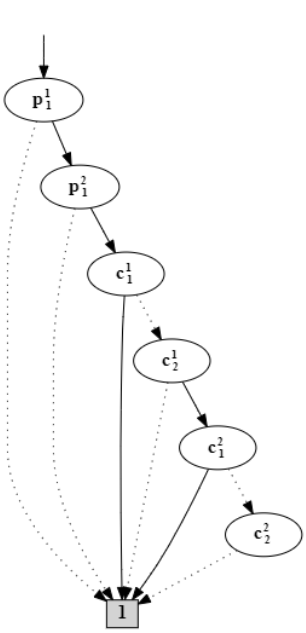
$P(d_1) = \pi_0$

$P(d_2) = \pi_3$

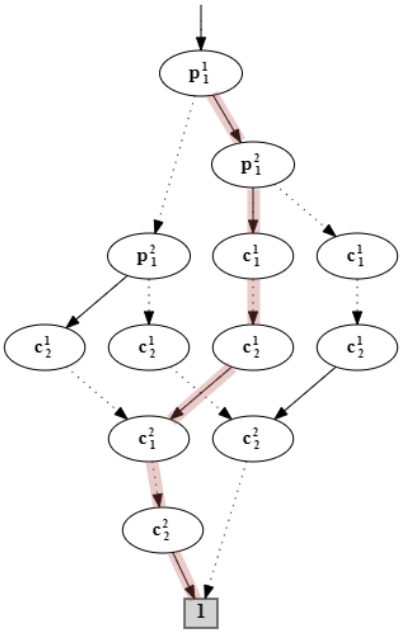
$\omega(d_1) = \{1\}$

$\omega(d_2) = \{1, 2\}$

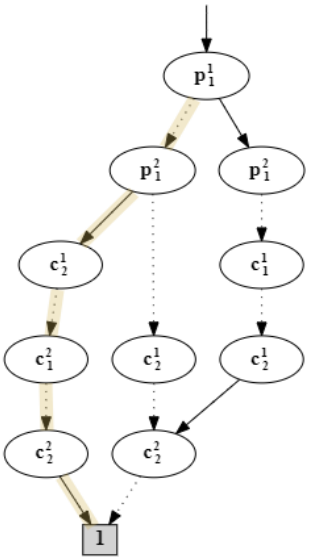
(a) Example of RSA problem and two corresponding solutions. Solution 2 is highlighted in yellow, and uses 2 slots. Solution 1 uses only 1 slot. In the case where the thick red edge fails, only solution 2 is valid.



(b) *noClash*.



(c) *assignment*.



(d) *rsa-all* = *assignment* ∧ *noClash*.

Fig. 1: BDDs *noClash*, *assignment*, and *rsa-all* for the example shown in Figure 1a. The red highlighted path in 1c shows the path assignments $P(d_1) = \pi_1$ and $P(d_2) = \pi_3$, and the channel assignments $\omega(d_1) = \omega(d_2) = \{1, 2\}$ in which the two demands can clash. It can be seen that these path and channel assignments have disappered in the BDD *rsa-all*. The yellow highlighted path in 1d corresponds to two solutions, one being Solution 2 from Figure 1a and the other being the same as Solution 2 except that $\omega(d_1) = \{2\}$.

failures occur. For this, we first examine how to use an adapted version of the ILP formulation from Miyagawa et al. [21] to ensure that we can handle any k-link failure scenario. We have chosen this specific ILP formulation as it is a state-of-the-art formulation that uses the same optimisation quality of minimizing the highest used slot in the network as we do. To adapt the ILP formulation, we simply relax the constraint from [21] that demands must always be bidirectional, as it is not a part of the RSA problem as defined in this paper. Otherwise, the ILP formulation is unmodified. The ILP formulation uses the integer variables $x_{d\pi f} \in \{0,1\}$ where $x_{d\pi f} = 1$ signifies that demand d uses path $\pi \in DPaths(d)$ with start index $f \in F$. Additionally, it uses the parameters $n_{d\pi}$ to denote the number of frequency slots demand d needs to be transmitted along path $\pi \in DPaths(d)$, i.e. $n_{d\pi} = size(d) \cdot \Delta(\pi)$. The ILP formulation is as follows:

$$minimize\, f_{max} \tag{1}$$

$$\sum_{f \in F} \sum_{\pi \in DPaths(d)} x_{d\pi f} = 1, \qquad \forall d \in D \tag{2}$$

$$\sum_{d \in D} \sum_{\substack{\pi \in DPaths(d) \\ e \in \pi}} \sum_{\substack{f' \in F \\ f - n_{d\pi} + 1 \leq f' \leq f}} x_{d\pi f'} \leq 1, \qquad \forall e \in E, \forall f \in F \tag{3}$$

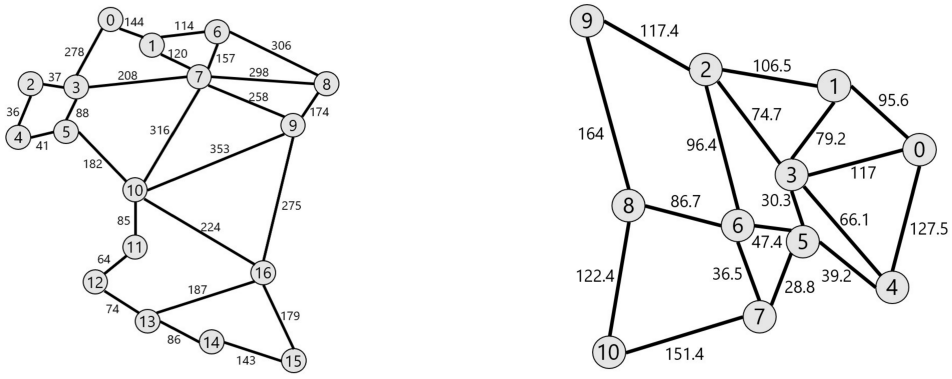
$$\sum_{f \in F} (f + n_{d\pi} - 1) \cdot x_{d\pi f} \leq f_{max}, \qquad \forall d \in D, \forall \pi \in DPaths(d) \tag{4}$$

Constraint 2 assigns a single path and start slot for each demand. Constraint 3 ensures that there is no clashing between demands on any edge of the paths. Finally, Constraint 4 ensures that the slots used by the demands are below f_{max}.

When link failures occur in the network, the ILP formulation can be used to compute a new solution that does not use the failed links. We evaluate this approach through an experiment, using the Gurobi ILP solver in Python [36], and run the experiment on a Ubuntu 18.04.5 cluster with 2.3 GHz AMD Opteron 6376 processors, with a memory limit of 30GB. For the experiment, we use the Deutsche Telecom Backbone (DT) and Kanto 11 network topologies (see Figure 2), wherein nodes have a population size, corresponding to the actual populations of the cities referenced from [37, 38, 39]. Using the population sizes, demands are generated using a gravity model as follows. First, a demand size is uniformly picked between 1 to 30, whereafter the source and target of the demand are chosen based on a population-based probability distribution. This process is repeated until $|D|$ demands are generated; see Appendix 1 for further details. Two shortest paths are generated for each demand using the semi-disjoint path generation algorithm from [40], and candidate channels are generated based on a 320 slot spectrum capacity [41] with modulation $\Delta(\pi) = 1$ for all $\pi \in \mathbf{Paths}$. Lastly, we simulate a k-link failure scenario by randomly failing k links in the network.

Figures 3a and 3b show box plots of the time it takes to find a new solution for 1-5 link failures for 3, 6, and 9 demands in the DT network and Kanto network respectively. We see that the ILP is incapable of finding a solution in less than 200 ms for 6 or 9 demands and only rarely for 3 demands in both networks. Hence, using the ILP formulation to compute a new solution when link failures occur is too slow to be used for a timely recovery of critical demands.

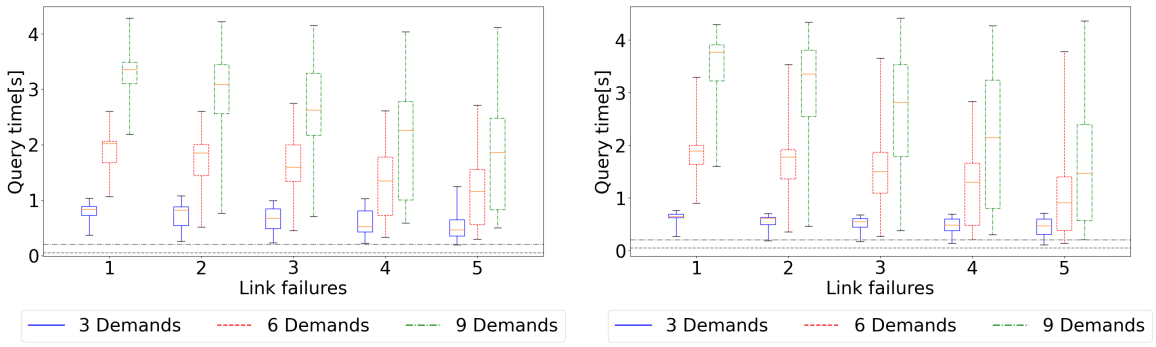
A different approach is to precompute all possible solutions for any k-or-fewer link failures such that a solution can be provided instantaneously when the link failures occur. Thus, for each possible combination of failed edges $E_{fail} \subseteq E$, where $|E_{fail}| \leq k$, we find a solution using the ILP formulation where no demand is assigned a path π that uses any edge $e \in E_{fail}$. Table 1 shows for 3, 6, and 9 demands how long it takes to compute for all $1 \leq k \leq 5$ possible link failures in the DT and Kanto networks. The computation time is measured by simulating 1000 k-link failures and then extrapolating the results to all $\frac{|E|!}{k!(|E|-k)!}$ k-link failure scenarios for a graph with edges E.



(a) Deutsche Telecom Backbone Network (DT) [42].

(b) Kanto 11 Network [42].

Fig. 2: Networks used for the evaluation. The numbers labeled on the edges indicate the distance between the nodes in kilometers.



(a) DT.

(b) Kanto.

Fig. 3: Boxplots of ILP query times based on 1000 random link-failure scenarios for the DT and Kanto 11 network topologies; lines are shown for 50 ms and 200 ms.

As seen in Table 1, k-link failures for $k > 3$ takes multiple days to compute in the DT network, and it takes close to or more than a day in the Kanto network for 6 or more demands. Hence, the problem with this approach is that the number of possible link failure scenarios grows quickly with regards to k, which means that the time to compute the required solutions also grows quickly. Additionally, the memory required to store all these solutions grows just as fast on the number of link failures. Precomputing solutions to link failures using ILP is thus not feasible and other approaches must be explored. In the next sections, we present our solution for efficiently computing and storing all solutions to a given RSA problem and how to leverage these to provide resilience against multiple link failures in a time-critical manner.

4 BDD Encoding of RSA

The first rendition of Binary Decision Diagrams (BDDs) was introduced by Lee [43] and Akers [44] as a data structure to efficiently represent and manipulate Boolean functions. The concept was later refined by Bryant [45] with more efficient Boolean operations. As such, BDDs are generally good at representing finite state problems [46] and have been applied in various problem domains [4, 5, 47,

Table 1: Extrapolated times for precomputing solutions for failure resilience using ILP.

DT/Kanto 11		Demands		
		3	6	9
Failures	1	42s/23s	2m/2m	3m/3m
	2	17m/7m	40m/19m	2h/35m
	3	5h/2h	11h/4h	17h/6h
	4	2d/10h	5d/23h	8d/2d
	5	19d/3d	39d/5d	60d/8d

Table 2: Variables used in the BDD encodings.

Variables	Description
$\overline{\mathbf{c}^d} = [\mathbf{c}^d_n, \mathbf{c}^d_{n-1}, ..., \mathbf{c}^d_1]$ where $n = \lceil \log_2(\lvert channels(d) \rvert) \rceil$	Binary encoding for channels in $channels(d)$
$\overline{\mathbf{p}^d} = [\mathbf{p}^d_n, \mathbf{p}^d_{n-1}, ..., \mathbf{p}^d_1]$ where $n = \lceil \log_2(\lvert DPaths(d) \rvert) \rceil$	Binary encoding for the paths in $DPaths(d)$

48, 49]. In this paper, we show how to apply BDDs to compute and represent all solutions to a given RSA problem and provide operations for finding new solutions when link failures occur.

Formally, a BDD is a directed, acyclic graph structure used to compactly represent Boolean functions [50]. Within a BDD, non-leaf nodes are labeled with Boolean variables, while leaf nodes are labelled with truth values 0 ($False$) or 1 ($True$). Each non-leaf node u has two outgoing edges denoted as $low(u)$ and $high(u)$ corresponding to its label variable being $False$ or $True$ respectively. These edges are commonly visualized by using a solid line for $high(u)$ and a dotted line for $low(u)$. A BDD is ordered (OBDD) if the variables in the BDD come in the same order $x_1 < x_2 < ... < x_n$ on all paths of the BDD [50]. An OBDD can be reduced by merging nodes with identical subgraphs, and by deleting nodes where the subgraphs for $low(u)$ and $high(u)$ are equivalent. Such a BDD is called a reduced OBDD (ROBDD) [50]. In the rest of the paper, we use the short form BDD in place of ROBDD. Lastly, we note that we use BDDs that support first-order quantifiers and that are closed under both Boolean operations and quantifiers [47].

4.1 Representing a set as a Boolean Function

Given a finite set $S = \{s_0, s_1, ..., s_{|S|-1}\}$, let $\overline{\mathbf{x}} = [\mathbf{x_k}, ..., \mathbf{x_1}]$ be a vector of Boolean variables where $k = \lceil log_2(|S|) \rceil$. Then, any truth assignment α to $\overline{\mathbf{x}}$ can be interpreted as a natural number $n(\alpha) \in \mathbb{N}$ written in binary notation. Thus $\overline{\mathbf{x}}$ encodes the $n(\alpha)$'th element of S. Let $\overline{\mathbf{x}}(s)$ denote the Boolean expression over $\overline{\mathbf{x}}$ with just the single truth assignment corresponding to $\{s\}$. Lastly, given some Boolean expression $b(\overline{\mathbf{x}})$, let $[\![b(\overline{\mathbf{x}})]\!]$ denote the encoded subset $\{s_{n(\alpha)} | \alpha \text{ satisfies } b(\overline{\mathbf{x}})\} \subseteq S$, such that $[\![b(\overline{\mathbf{x}})]\!]$ is a set consisting of the elements, that are encoded by the Boolean assignments which satisfy the Boolean expression b.

Example 3. Considering the set of paths $\{\pi_0, \pi_1, \pi_2, \pi_3\}$ from Figure 1a, we need two boolean variables $\overline{\mathbf{p}} = [\mathbf{p_2}, \mathbf{p_1}]$ to encode any of the given paths. For instance, we encode the path $\{\pi_0\}$ by $\overline{\mathbf{p}}(\pi_0) = \neg\mathbf{p_2} \wedge \neg\mathbf{p_1}$. In a Boolean expression such as $b = \mathbf{p_1}$, where the variable $\mathbf{p}_2$ is free, the Boolean function $b(\overline{\mathbf{p}})$ is satisfied both when $[\mathbf{p_2} \mapsto 0, \mathbf{p_1} \mapsto 1]$ and $[\mathbf{p_2} \mapsto 1, \mathbf{p_1} \mapsto 1]$, which means $[\![b(\overline{\mathbf{p}})]\!] = \{\pi_1, \pi_3\}$.

4.2 Construction of BDDs

As shown in Table 2, for each demand d, we use a vector of Boolean variables $\overline{\mathbf{p}^d}$ to encode the path assigned to demand d, and another vector of Boolean variables $\overline{\mathbf{c}^d}$ to encode the channel assigned to

demand d. To encode a solution (P, ω) to the RSA problem described in Definition 1, we use the vector of vectors $\overline{\mathbf{p}^D} = [\overline{\mathbf{p}^{d_1}}, \overline{\mathbf{p}^{d_2}}, ..., \overline{\mathbf{p}^{d_{|D|}}}]$ to encode P and the vector of vectors $\overline{\mathbf{c}^D} = [\overline{\mathbf{c}^{d_1}}, \overline{\mathbf{c}^{d_2}}, ..., \overline{\mathbf{c}^{d_{|D|}}}]$ to encode ω.

Example 4. We can see that the BDD *assignment* shown in Figure 1c represents all $3^2 = 9$ possible combinations of P and ω for the running example. As an example of how to interpret the BDD, we see that the specific assignment $P(\{d_1, d_2\}) = \{\pi_1, \pi_3\}$ and $\omega(\{d_1, d_2\}) = \{\{1, 2\}, \{1, 2\}\}$ is highlighted in red.

We now define three BDDs which enforce that $\overline{\mathbf{p}^D}$ and $\overline{\mathbf{c}^D}$ must only encode valid solutions to the RSA problem. Specifically, the BDDs must ensure that each demand d is assigned a path $\pi \in DPaths(d)$ and a channel $C \in channels(d)$ such that $|C| = \Delta(\pi) \cdot size(d)$, and that, whenever a demand shares an edge with another demand, they cannot be assigned overlapping channels.

To enforce that no two demands are allowed to clash, we define the BDD *noClash* (Figure 1b) as

$$\textit{noClash}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \bigwedge_{\substack{d,d' \in D,\\ d \neq d'}} \bigwedge_{\substack{\pi \in DPath(d),\\ \pi' \in DPaths(d'),\\ \pi \cap \pi' \neq \emptyset}} \left(\neg\big(\overline{\mathbf{p}^d}(\pi) \wedge \overline{\mathbf{p}^{d'}}(\pi')\big) \vee \left(\bigwedge_{\substack{C \in channels(d),\\ C' \in channels(d'),\\ |C| = \Delta(\pi) \cdot size(d),\\ |C'| = \Delta(\pi') \cdot size(d'),\\ C \cap C' \neq \emptyset}} \neg\big(\overline{\mathbf{c}^d}(C) \wedge \overline{\mathbf{c}^{d'}}(C')\big) \right) \right) \quad (5)$$

that satisfies $(P, \omega) \in [\![\textit{noClash}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ iff for all $d, d' \in D$ where $d \neq d'$ either $P(d) \cap P(d') = \emptyset$ or $\omega(d) \cap \omega(d') = \emptyset$.

Example 5. Consider again the network from Figure 1a. There is only one way the two demands can clash. The BDD *noClash* therefore has to encode that it is satisfied by all path and channel assignments, except when d_1 and d_2 are assigned the paths π_1 and π_3 respectively, and the channels $\omega(d_1) = \omega(d_2) = \{1, 2\}$. As shown in Figure 1b, the BDD *noClash* encodes exactly this, as the only way not to satisfy this BDD is taking the right-most path down to the node labeled $\mathbf{c}_2^2$, and then setting the value of the Boolean variable $\mathbf{c}_2^2$ to $True$, which corresponds to the clashing assignment.

Having defined the BDD *noClash*, we must enforce that each demand d is assigned a correct path and channel assignment pair. To this end, we define the BDD *assignment* as

$$\textit{assignment}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \bigwedge_{d \in D} \bigvee_{\pi \in DPath(d)} \left(\overline{\mathbf{p}^d}(\pi) \wedge \Big(\bigvee_{\substack{C \in channels(d),\\ |C| = \Delta(\pi) \cdot size(d)}} \overline{\mathbf{c}^d}(C) \Big) \right) \quad (6)$$

and clearly, $(P, \omega) \in [\![\textit{assignment}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ iff for all $d \in D$, it holds that $P(d) \in DPaths(d)$, $\omega(d) \in channels(d)$ and $|\omega(d)| = \Delta(P(d)) \cdot size(d)$.

Finally, we use *assignment* and *noClash* to define the BDD *rsa-all* as

$$\textit{rsa-all}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \textit{assignment}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \wedge \textit{noClash}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}). \quad (7)$$

Example 6. For the running example in Figure 1a, we see in Figure 1d how *rsa-all* contains all valid solutions for the given RSA problem. All the assignments from the BDD *assignments* are valid

solutions, except the assignment that is marked red in Figure 1c, since it is the only assignment that does not satisfy the BDD *no-clash*, as noted in Example 5. Hence, this is the only assignment that does not appear in *rsa-all*. The highlighted path in *rsa-all* corresponds to two valid solutions, one solution being Solution 2 from Figure 1a, the other being the same as Solution 2 except that demand d_1 is assigned channel $\{2\}$ instead of channel $\{1\}$.

Theorem 1. *The pair (P,ω) is a solution to the RSA problem iff $(P,\omega) \in [\![\mathit{rsa\text{-}all}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$.*

Proof. "$\Longrightarrow$" Assume (P,ω) is a solution to the RSA problem. By Definition 1, the encodings $\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}$ of (P,ω) satisfy both the constraints enforced by *assignment* and *noClash*, and it thus follows that $(P,\omega) \in [\![\mathit{rsa\text{-}all}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$.

"$\Longleftarrow$" Let $(P,\omega) \in [\![\mathit{rsa\text{-}all}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$. By the constraints enforced by *assignment* we know that each demand has been assigned a valid path and channel combination given the used modulation, and due to the constraints enforced by *noClash* we know that the path and channel assignments given to each demand result in a solution, with no clashing between any of the demands. Hence (P,ω) is a solution to the RSA problem. □

With the representation of all solutions in $\mathit{rsa\text{-}all}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})$, it is possible for a network operator to query it to find for example an optimal solution or a solution matching any particular properties desired by the operator.

For the k-link failover problem, there is no need to have all solutions represented, as we need only to represent the optimal path and channel assignment for each possible scenario of up to k link failures. To limit the set of solutions represented, we first see that the solution space is bloated with channel assignments that are fragmented. A fragmented channel assignment contains gaps of slots between the assigned channels, and intuitively, an optimal solution cannot contain these gaps. Thus, to remove gaps, we require that a channel assignment ω must assign all slots below $usage(\omega)$ to at least one demand. This is enforced by specifying that the channel assigned to a demand either uses the first slot, or follows directly after a channel assigned to another potentially path-overlapping demand.

Definition 3 (Gap-free). *A solution to the RSA problem (P,ω) is* gap-free *if for all $d \in D$, either $min(\omega(d)) = 1$ or there exists a $d' \in D, \pi' \in DPaths(d')$ and $\pi \in DPaths(d)$ s.t. $\pi \cap \pi' \neq \emptyset$ and $min(\omega(d)) = max(\omega(d')) + 1$.*

The gap-free property can be imposed on the solutions encoded by the BDD *rsa-all* using the BDD *gapfree*, which is defined as

$$\mathit{gapfree}(\overline{\mathbf{c}^D}) = \tag{8}$$

$$\bigwedge_{d \in D} \left(\Big(\bigvee_{\substack{C \in channels(d),\\ min(C)=1}} \overline{\mathbf{c}^d}(C) \Big) \vee \Big(\bigvee_{\substack{d' \in D,\\ \exists \pi \in DPaths(d),\\ \exists \pi' \in DPaths(d'),\\ \pi \cap \pi' \neq \emptyset}} \; \bigvee_{\substack{C \in channels(d),\\ C' \in channels(d'),\\ min(C)=max(C')+1}} \overline{\mathbf{c}^d}(C) \wedge \overline{\mathbf{c}^{d'}}(C') \Big) \right) \tag{9}$$

and we have $\omega \in [\![\mathit{gapfree}(\overline{\mathbf{c}^D})]\!]$ iff ω satisfies the gap-free channel property as described in Definition 3. Additionally, since the gap-free property preserves an optimal solution for each routing assignment, it can handle the same link failure scenarios as *rsa-all*.

Theorem 2. *If (P,ω) is a solution to an RSA problem, then there exists a gap-free solution (P,ω'), s.t. $usage(\omega') \leq usage(\omega)$.*

Proof. Assume a channel assignment ω for demands $d_1, d_2, ..., d_m$ such that $d_i \leq d_j$ if $min(\omega(d_i)) \leq min(\omega(d_j))$. A new ω' can now be constructed, such that ω' is *gap-free*. For d_i, let D_i be the set of demands $d_k < d_i$ where $\omega(d_k) \cap \omega(d_i) = \emptyset$ and there exists $\pi_k \in DPaths(d_k)$ and $\pi_i \in DPaths(d_i)$ s.t. $\pi_k \cap \pi_i \neq \emptyset$. Let ω' initially be undefined for all demands. We now add each demand to ω' one by one in the specified order, such that if $D_i = \emptyset$ then $\omega'(d_i) = \{1, ..., |\omega(d_i)|\}$, otherwise let the highest slot assigned by ω' to a demand in D_i be $\Omega = \underset{d \in D_i}{max}(max(\omega'(d)))$ and then the new channel of d_i is specified as $\omega'(d_i) = \{\Omega + 1, \Omega + 2, ..., \Omega + |\omega(d_i)|\}$. It follows that $usage(\omega') \leq usage(\omega)$ since at any step $min(\omega'(d_i)) \leq 1 + |\bigcup_{d \in D_i} \omega(d)|$. □

We now define a new BDD $\mathit{rsa\text{-}gapfree}$ using $\mathit{gapfree}$ to reduce the number of encoded solutions compared to $\mathit{rsa\text{-}all}$. The BDD $\mathit{rsa\text{-}gapfree}$ is defined as

$$\mathit{rsa\text{-}gapfree}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \mathit{assignment}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \wedge \mathit{gapfree}(\overline{\mathbf{c}^D}) \wedge \mathit{noClash}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \tag{10}$$

and clearly $(P, \omega) \in [\![\mathit{rsa\text{-}gapfree}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ iff (P, ω) is a valid solution to the RSA problem as described in Definition 1, and ω satisfies the gap-free property. Since $\mathit{assignment}$, $\mathit{gapfree}$ and $\mathit{noClash}$ are just Boolean expressions, their order in the three-way conjunction is semantically irrelevant. In the implementation, the conjunction is evaluated in a left-to-right order, since this results in the best performance.

In addition to removing gaps, any optimal solution requires at most a spectrum width of $max_f = \sum_{d \in D} \underset{\pi \in DPaths(d)}{max} \big(\Delta(\pi) \cdot size(d)\big)$ and thus all channels where $max(C) > max_f$ can be disregarded. By enforcing this upper limit, we effectively prune the candidate channels for each demand without losing optimality. A more greedy approach to approximate the candidate channels is to assign channels based on an established ordering of the demands. The idea is to remove symmetry in the channel assignments by imposing the property of *limited* as defined in Definition 4; the property is imposed on the mapping *channels* in the definition of $\mathit{gapfree}$ in Equation 9.

Definition 4 (Limited). *Given demands $D = \{d_1, d_2, ..., d_m\}$, a solution to the RSA problem (P, ω) is* limited *if $min(\omega(d_i)) \leq c_{max} + \sum_{j<i} |\omega(d_j)|$ for all $d_i \in D$, where $c_{max} = \underset{d \in D}{max} |\omega(d)|$.*

We note that the greedy approach will in some instances make it impossible to find a channel assignment for a particular path assignment (see counter example in Appendix 2). This means that a BDD with this property imposed cannot generally be said to contain an optimal solution for all link failure scenarios. However, from running a trial on all topologies in the standard benchmark topology collection, Topology Zoo [51], with demands ordered from largest to smallest, we did not find any instance where a counter example was encountered. We therefore argue that for all intents and purposes of this paper that the property *limited* preserves an optimal solution for each link failure scenario.

Henceforth, we will use $\mathit{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})$ to refer to any BDD that represents only valid solutions to a given RSA problem.

4.3 Finding the Optimal Solution

Once we have a BDD representing only valid solutions to an RSA problem, we are interested in finding an optimal solution. To this end, we introduce a new vector of variables $\overline{\mathbf{s}^F} = [\mathbf{s}_1, \mathbf{s}_2, ...\mathbf{s}_{f_{max}}]$. We say that $\overline{\mathbf{s}^F}$ encodes a subset $F' \subseteq F$ s.t. $\mathbf{s}_f$ is true iff $f \in F'$.

The intuition behind the use of these variables is that if a variable $\mathbf{s}_f$ is false, then we know that no demand has been assigned a channel with a slot greater than or equal to f, meaning that the usage is at most $f - 1$. We can enforce this using the BDD $\mathit{rsa\text{-}slotBound}$ defined as

$$\texttt{rsa-slotBound}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{s}^F}) \tag{11}$$

$$= \texttt{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \wedge \left(\bigwedge_{d \in D} \bigvee_{C \in channels(d)} \left(\overline{\mathbf{c}^d}(C) \wedge \bigwedge_{f \leq max(C)} \mathbf{s}_f \right) \right) \tag{12}$$

and clearly $(P, \omega, F') \in [\![\texttt{rsa-slotBound}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{s}^F})]\!]$ iff $(P, \omega) \in [\![\texttt{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ and F' contains all $f \leq usage(\omega)$.

An optimal solution can now be found by identifying the smallest value of $f \in F$ such that there exists a solution $(P, \omega, F') \in [\![\texttt{rsa-slotBound}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{s}^F})]\!]$ where the corresponding variable $\mathbf{s}_f$ can take the value false, i.e. $f \notin F'$, since it can then be inferred that the $usage(\omega) = f - 1 \leq usage(\omega')$ for all other $(P', \omega', F'') \in [\![\texttt{rsa-slotBound}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{s}^F})]\!]$.

5 Using BDDs to Ensure Failure Resilience

Having defined the BDD `rsa` to find solutions to the RSA problem, we now show how to extract solutions from the BDD for the purpose of providing time-critical responses to multiple links failing. In particular, the BDD `rsa` as defined in Section 4.2 contains at least one solution for all failure scenarios, if such a solution is possible. Furthermore, we know that at least one of these solutions is optimal under the corresponding failure scenario. To extract these solutions, we present two approaches. The first method supports an arbitrary number of link failures based on the idea of deleting solutions that use invalid paths given the link failures. The second method uses pre-computation in which link failures are directly encoded in the BDD when constructed, which allows for more efficient path pruning times.

5.1 Finding Failover Solutions

Pruning by Deletion Solutions encoded in the BDD `rsa` that use invalid paths based on the set of link failures E_{fail} are no longer valid solutions. The invalid solutions must therefore be deleted, which we do using the BDD `path-pruned-rsa` defined as

$$\texttt{path-pruned-rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \texttt{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \wedge \bigwedge_{d \in D} \bigwedge_{\substack{\pi \in DPaths(d), \\ \exists e \in E_{fail}, \\ e \in \pi}} \neg \overline{\mathbf{p}^d}(\pi) \tag{13}$$

and clearly $(P, \omega) \in [\![\texttt{path-pruned-rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ iff $(P, \omega) \in [\![\texttt{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ and it holds for all $d \in D$ that $P(d) \cap E_{fail} = \emptyset$. Since there is no limitation on the size E_{fail}, this pruning method can be used to prune the BDD for any $0 \leq k \leq |E|$ link failures.

Pruning by Precomputation Accounting for larger k link failures generally gives diminishing returns, since the risk of that many links failing at the same time becomes increasingly small. We can thus use this to our advantage by constructing a parameterized BDD with additional variables representing the failed edges; these can be specified to retrieve the valid solutions for any given link failure scenario up to some sufficiently large k.

We define the new set $E_u = E \cup \{e_{unused}\}$ where the auxiliary edge e_{unused} signifies a non-failed link. Then, for a given k-link failover problem, we introduce a vector of k variable encodings $\overline{\mathbf{e}^K} = [\overline{\mathbf{e}^1}, ..., \overline{\mathbf{e}^k}]$ such that $\overline{\mathbf{e}^i} = [\mathbf{e}^i_n, \mathbf{e}^i_{n-1}, ..., \mathbf{e}^i_1]$ for $i \in K = \{1, 2, ..., k\}$, where $n = \lceil \log_2(|E_u|) \rceil$. The variable $\overline{\mathbf{e}^i}$ thus either encodes a specific link-failed edge or the auxiliary edge e_{unused} signifying

that it does not encode any link failing. This makes it possible to encode up to k link failures, rather than being limited to exactly k links failing.

We can now define the BDD $\mathit{path\text{-}edge\text{-}overlap}$ to associate each path with its constituent edges

$$\mathit{path\text{-}edge\text{-}overlap}(\overline{\mathbf{p}^D}, \overline{\mathbf{e}}) = \bigvee_{d \in D} \bigvee_{\pi \in DPaths(d)} \bigvee_{e \in \pi} \overline{\mathbf{p}^d}(\pi) \wedge \overline{\mathbf{e}}(e) \tag{14}$$

and clearly $(P, e) \in [\![\mathit{path\text{-}edge\text{-}overlap}(\overline{\mathbf{p}^D}, \overline{\mathbf{e}})]\!]$ iff there exists a $\pi \in P(D)$ such that $e \in \pi$.

Using this, we define the BDD $\mathit{failover}^k$ to encode the valid path assignments for every combination of k or fewer link failures as

$$\mathit{failover}^k(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{e}^K}) = \mathit{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \wedge \tag{15}$$

$$\bigvee_{\substack{E' = \{e_1, e_2, \ldots, e_m\} \subseteq E,\ 1 \le i \le m \\ |E'| \le k}} \bigwedge \left(\overline{\mathbf{e}^i}(e_i) \wedge \neg \mathit{path\text{-}edge\text{-}overlap}(\overline{\mathbf{p}^D}, \overline{\mathbf{e}^i})\right) \wedge \bigwedge_{m < j \le k} \overline{\mathbf{e}^j}(e_{unused}). \tag{16}$$

Theorem 3. *Let $E_{fail} \subseteq E$ be a subset of failed edges where $|E_{fail}| \le k$, and let the vector of variable encodings $\overline{\mathbf{e}^K}$ encode the set E_{fail}. Then, $(P, \omega, E_{fail}) \in [\![\mathit{failover}^k(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{e}^K})]\!]$ iff $(P, \omega) \in [\![\mathit{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ is a solution to the k-link failover problem for link failures E_{fail}.*

Proof. "$\Longrightarrow$" Let a solution be $(P, \omega, E_{fail}) \in [\![\mathit{failover}^k(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{e}^K})]\!]$ where $|E_{fail}| \le k$. From Condition 15, we know that $(P, \omega) \in [\![\mathit{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ and thereby is a valid solution to the RSA problem. Furthermore, as $\overline{\mathbf{e}^K}$ encodes E_{fail}, we know that for every edge e in E_{fail} that there exists a variable encoding $\overline{\mathbf{e}^i}$ from $\overline{\mathbf{e}^K}$ that encodes e. Additionally, we know that for all e in E_{fail} that $(P, e) \notin [\![\mathit{path\text{-}edge\text{-}overlap}(\overline{\mathbf{p}^D}, \overline{\mathbf{e}^i})]\!]$ and thus none of the paths assigned to the demands contain any of the failed edges, which means that (P, ω, E_{fail}) is a solution.

"$\Longleftarrow$" Let $(P, \omega) \in [\![\mathit{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ be a solution to the k-link failover problem for a set of failed edges $E_{fail} \subseteq E$ where $|E_{fail}| \le k$, and demands D. By Definition 2, we know that for all $\pi \in P(D)$ that $\pi \cap E_{fail} = \emptyset$. Hence, the BDD $\neg \mathit{path\text{-}edge\text{-}overlap}(\overline{\mathbf{p}^D}, \overline{\mathbf{e}^i})$ is satisfied for all e. As such, Condition 16 is satisfied and $(P, \omega, E_{fail}) \in [\![\mathit{failover}^k(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{e}^K})]\!]$. □

Using the BDD $\mathit{failover}^k$, we define the BDD $\mathit{path\text{-}pruned\text{-}rsa}$ which encodes the valid solutions given a set of specific link failures E_{fail} where $|E_{fail}| \le k$

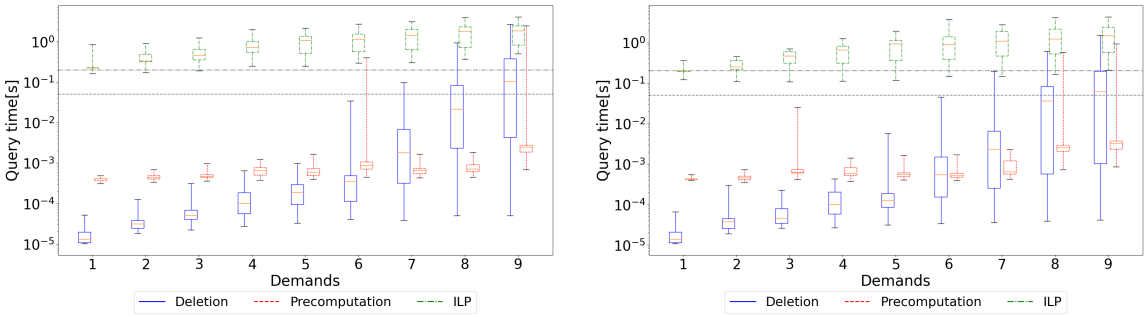
$$\mathit{path\text{-}pruned\text{-}rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \tag{17}$$

$$\exists \overline{\mathbf{e}^K}. \Big(\bigwedge_{e_i \in E_{fail} = \{e_1, e_2, \ldots, e_m\}} \overline{\mathbf{e}^i}(e_i) \Big) \wedge \Big(\bigwedge_{m < j \le k} \overline{\mathbf{e}^j}(e_{unused}) \Big) \wedge \mathit{failover}^k(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}, \overline{\mathbf{e}^K}) \tag{18}$$

and clearly $(P, \omega) \in [\![\mathit{path\text{-}pruned\text{-}rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ iff $(P, \omega) \in [\![\mathit{rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ and it holds for all $d \in D$ that $P(d) \cap E_{fail} = \emptyset$, where $|E_{fail}| \le k$. Note that we reuse the name $\mathit{path\text{-}pruned\text{-}rsa}$ from Equation 13 since the BDDs represent the exact same solutions for the given failure scenario.

5.2 Finding Lightpath-preserving Solutions

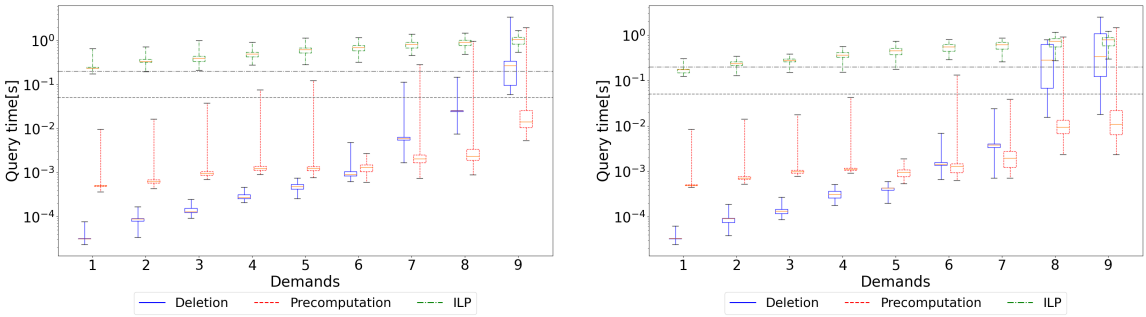
A property that is often desirable when finding a solution to a k-link failover problem, is that the new path and channel assignment should only differ from their old counterparts on the demands, that are directly affected by the link failures. The reason for this is to minimize the number of physical changes that must be performed in the network during the failover operation. We call this type of solution *lightpath-preserving*.



(a) Results for the DT topology.

(b) Results for the Kanto topology.

Fig. 4: The time it takes to query for an optimal solution for 5 link failures. The boxplot is drawn based on 1000 queries. Lines are shown for 50 ms and 200 ms.



(a) Results for the DT topology.

(b) Results for the Kanto topology.

Fig. 5: The time it takes to query for an optimal lightpath-preserving solution for 5 link failures. The boxplot is drawn based on 1000 queries. Lines are shown for 50 ms and 200 ms.

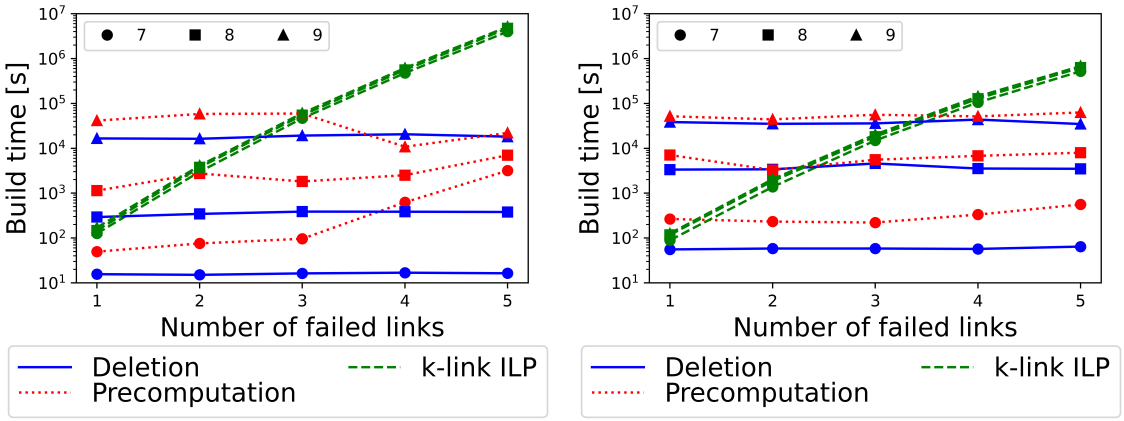
Definition 5 (Lightpath-preserving). *A solution* (P', ω') *is a* lightpath-preserving *solution to the RSA solution* (P, ω) *for link failures* E_{fail} *iff* $P(d) \cap E_{fail} = \emptyset \implies P(d) = P'(d) \wedge \omega(d) = \omega'(d)$ *for all* $d \in D$ *and* (P', ω') *is a solution for the k-link failover problem for* E_{fail}.

Given a solution (P, ω) to a given RSA problem, we find lightpath-preserving solutions in any BDD that encodes the valid solutions to said RSA problem, using the BDD $\mathit{rsa\text{-}lightpath\text{-}preserving}$, which is defined as

$$\mathit{rsa\text{-}lightpath\text{-}preserving}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \tag{19}$$

$$\mathit{path\text{-}pruned\text{-}rsa}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) \wedge \Big(\bigwedge_{\substack{d \in D, \\ P(d) \cap E_{fail} = \emptyset}} \overline{\mathbf{p}^d}(P(d)) \wedge \overline{\mathbf{c}^d}(\omega(d)) \Big) \tag{20}$$

and $(P', \omega') \in [\![\mathit{rsa\text{-}lightpath\text{-}preserving}(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ iff (P', ω') is a *lightpath-preserving* solution to the given RSA solution (P, ω) for link failures E_{fail}.



(a) Results for the DT topology with 52 links.

(b) Results for the Kanto topology with 36 links.

Fig. 6: The plots show how the number of link failures affects the time it takes to prepare the BDDs used for querying using the approaches *Deletion* and *Precomputation* respectively compared to how long it takes to compute k-link resilient solutions using the ILP approach described in Section 3. The different symbols represents the results with 7,8 or 9 demands respectively.

6 ExpectAll Implementation and Evaluation

We implemented the BDD encoding of the RSA problem presented in Section 4.2 and the two approaches for solving the k-link failover problem from Section 5 in the tool ExpectAll, using all the improvements suggested in Section 4.2. We implemented ExpectAll in Python using a Cython wrapper [52] of the library CUDD [53] to perform BDD operations. The source code of ExpectAll and our experimental artefacts, allowing for the experiments to be rerun, is available in [14].

We now evaluate the two BDD approaches to the k-link failover problem against the ILP approach from Section 3. The BDD approach using path-deletion pruning is called *deletion*, and the one using precomputation pruning is called *precomputation*. We compare their query times for finding an optimal solution and an optimal lightpath-preserving solution, by simulating 1000 random k-link failure scenarios. Results are shown in Figures 4 and 5 respectively for 5 link failures, with additional results for 1-4 link failures in Appendix 4 and 5.

It can be seen in Figure 4 that both BDD approaches significantly outperform the ILP model in the time it takes to find an optimal solution. The query time using the *deletion* method, however, increases with the number of demands, exceeding the 200 ms threshold at 8-9 demands, which indicates limited scalability. The *precomputation* method in contrast has lower variance and maintains query times well below 50 ms for most scenarios, making it highly reliable for rapid recovery.

Furthermore, the results in Figure 5 show that while the ILP approach is slightly faster at finding an optimal lightpath-preserving solution than a general optimal solution, both BDD approaches are still able to find an optimal lightpath-preserving solution faster than the ILP approach for seven or fewer demands and the *precomputation* approach remains consistently under 50 ms for up to nine demands. It is important to note that the experiments are performed on BDDs with all three improvements mentioned in Section 4.2. Thus, not all solutions to the RSA problem are represented in the BDD, which means that some lightpath-preserving solutions might be missed. Our findings show that as the number of demands increases, the risk of missing a lightpath-preserving solution also increases. However, for scenarios with four or fewer demands, we consistently find a lightpath-preserving solution. Full details of this are found in Appendix 3.

6.1 Measuring Build Time

Comparing the time it takes to find a single solution when k links fail, we see that our BDD based approaches outperform the ILP approach. However, we must also evaluate how long it takes to build the underlying BDDs in order to understand their applicability. We thus measure the time it takes to build the BDDs used for querying, which we compare with the computation time required for computing an optimal solution for every link failure scenario using ILP as described in Section 3. We performed the experiment on the same machine used for the previous experiment with a timeout of 24 hours. Within this timeout we could build BDDs for RSA problems with up to nine demands in the DT and Kanto network topology. For the ILP we estimate the build time, as described in Section 3.

The results, presented in Figure 6, indicate that the ILP computation time grows exponentially with the number of failed links. In contrast, the build time for the *deletion* method remains nearly constant because the BDD always represents the same number of solutions, rendering the method independent of the number of link failures. The build time of the *precomputation* method increases with the number of link failures, especially with fewer demands. As the number of demands increases, the time to construct the basic BDD becomes significantly larger, diminishing the impact of the marginal increase in build time due to additional link failures. Therefore, for a higher number of demands, the scalability issue of the *precomputation* method with regards to link failures becomes less pronounced.

Finally, the results in Figure 6 indicate that the threshold value k, beyond which the BDD-based approaches become advantageous to the ILP approach, depends on both the number of demands and the complexity of the network topology in terms of its number of links.

7 Conclusion

We have presented *ExpectAll*, a tool that can efficiently compute and represent *all* possible solutions through Binary Decision Diagram (BDD) technology; a task that is practically infeasible with state-of-the-art tools relying on Integer Linear Programming (ILP) to find singular solutions. Moreover, we demonstrated how this comprehensive representation of solutions can be applied to provide resilience against an arbitrary number of concurrent link failures for critical demands.

Our experiments with the prototype implementation of the tool *ExpectAll* demonstrate its capability to facilitate network recovery within well-established time frames. This shows that our approach can be used as a novel method for enhancing network survivability. By eliminating the necessity to establish and allocate resources for backup lightpaths solely for specific link failure scenarios, our method reduces redundancy and optimizes resource usage. It is important to note here, that our work only concerns a subset of demands in the network, which can be classified as critical. This means that some work must be done to determine how all the remaining demands in the network should be handled. For this, we propose, that these non-critical demands can be scheduled using a heuristic method that places them as far towards the end of the spectrum as possible. This way we minimize the risk of critical demands channel overlapping with the non-critical demands during any k-link failure scenario.

There are two primary directions for future research to broaden the applicability of our work: (1) Developing techniques to represent solutions for RSA problems with a larger number of demands using BDDs will be crucial for practical deployment; (2) Generalizing the methods for finding optimal and lightpath-preserving solutions will be highly beneficial. This involves constructing and querying BDDs to support efficient searches for solutions with any properties expressible as Boolean expressions. Such advancements will enhance the flexibility and applicability of our tool across various

networks and requirements.

Bibliographic Remark We note that the general introduction of BDDs in Section 4 and the definition of the binary set encoding in Subsection 4.1 are slightly modified versions of the corresponding sections in our pre-specialization work, that was defended in January of 2024 [54].

Epilogue

From our findings of applying BDDs to support optimal failover protection, it is evident that improving the scalability of the BDDs is an area of interest in order to be able to handle a larger number of critical demands in a network. In this secondary part of our work, we present our findings on improving the scalability of the presented BDDs. We present two of the methods that we have explored and discuss how these methods can not be part of our primary work, as they require that we make comprises in regards to efficient resource usage or ensuring k-link failure resilience.

Clique-bound Channels

We are able to further limit the candidate channels for each demand, by estimating an upper bound for what slots the channels can have, since it is known what other demands the demand may possibly share an edge with. Let Γ be a graph where the nodes represent a demand d in D. Given two distinct demands $d, d' \in D$, add an edge between the two nodes in Γ if there exists two paths $\pi^d \in DPaths(d)$ and $\pi^{d'} \in DPaths(d')$, and the two paths overlap on any edge. Based on Γ, we can find cliques, where cliques represent subsets of demands that potentially overlap with one another on their paths. Let $maxClique : D \to 2^D$ represent the maximum clique in Γ which demand d is part of.

Definition 6 (Clique-bound). *A solution to the RSA problem* (P, ω) *is* clique-bound *iff for all* $d \in D$, $max(\omega(d)) \leq \sum_{d' \in maxClique(d)} \max_{C \in channels(d')} |C|$.

It must be noted that the property *clique-bound* in some circumstances will find that no channel assignment is possible for a path assignment, even though a valid non *clique-bound* channel assignment does exist. An example of this occurs if some of the paths always overlap such that they form a Mycielski graph in Γ [55].

We can combine the *clique-bound* property with the *limited* property. First, we introduce the mapping *cliques* such that $cliques(d)$ is the set of all cliques that demand d is a part of, using the same definition of a clique as the mapping $maxClique$. Then, for each demand d, we assume an ordering on the demands in $maxClique(d)$ such that for $d', d'' \in maxClique(d), d' \leq d''$ if $|cliques(d')| \leq |cliques(d'')|$. Based on this ordering, we enforce the *limited* property on the largest clique of each demand. If a demand is part of more than one of the largest cliques, we pick the limited constraint that allows the most channel assignments for that demand.

Definition 7 (Limited Clique-bound). *A solution to the RSA problem* (P, ω) *is* limited clique-bound *iff for all* $d \in D$, $min(\omega(d)) \leq \max_{\substack{d' \in D, \\ d \in maxClique(d')}} \sum_{\substack{d'' \in maxClique(d'), \\ d'' < d}} \max_{C \in channels(d'')} |C|$.

Subspectrums

We can split a given RSA problem into n RSA subproblems by splitting the spectrum of slots F into n disjoint subspectrums, i.e. $F = \bigcup_{i=1}^{n} F_i$, and the demands D into n disjoint sets of demands, i.e. $D = \bigcup_{i=1}^{n} D_i$. Thus, a demand $d \in D_i$ may only use the channels contained in its subspectrum such that $C \in channels(d)$ iff $min(F_i) \leq min(C)$ and $max(C) < max(F_i)$. Using this approach, each RSA subproblem can be solved independently, decreasing the complexity of the original RSA problem at the cost of a higher usage of the spectrum. We solve the RSA subproblem for D_i and

F_i by computing the BDD rsa^i for all $1 \leq i \leq n$ and then conjugate all these BDDs into one BDD $\mathit{rsa\text{-}sub}$ which encodes solutions to the original RSA problem, i.e.

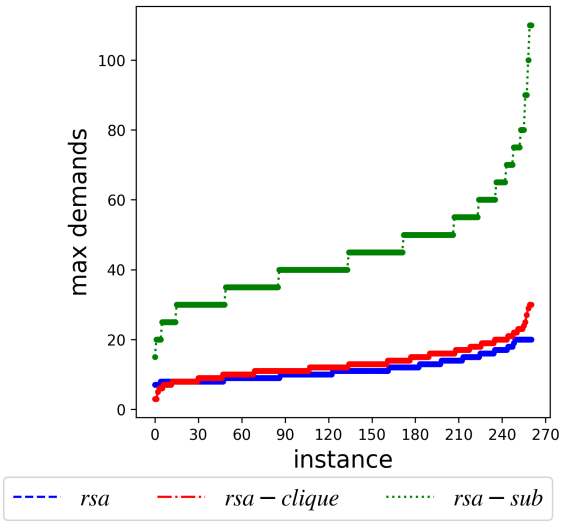
$$\mathit{rsa\text{-}sub}\,(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D}) = \bigwedge_{i \leq n} \mathit{rsa}^i(\overline{\mathbf{p}^{D_i}}, \overline{\mathbf{c}^{D_i}}) \tag{21}$$

and clearly, if $(P, \omega) \in [\![\mathit{rsa\text{-}sub}\,(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$ then $(P, \omega) \in [\![\mathit{rsa}\,(\overline{\mathbf{p}^D}, \overline{\mathbf{c}^D})]\!]$, as a channel assignment encoded by rsa^i can never have overlapping channels with a channel assignment from $\mathit{rsa}^j, i \neq j$. We note that the performance of $\mathit{rsa\text{-}sub}$ depends on the number of splits n as well as the distribution of slots and demands among the splits. Currently, we split the spectrum F into k equally sized subspectrums, adding any remaining slots to the last subspectrum F_n if an even split is not possible. We then assign each demand a subspectrum sequentially based on how the given demand can overlap on its paths with other demands that have already been assigned a subspectrum. Specifically, we place a given demand $d \in D_i$ if D_i contains the fewest number of demands which d can overlap with on one of its paths. Additionally, if $n > |D|$, then we only make $|D|$ splits.

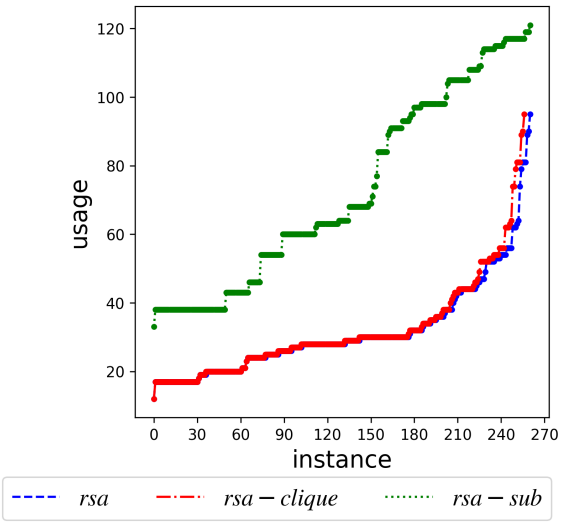
Understanding the Trade-offs

The methods have been implemented in the BDDs $\mathit{rsa\text{-}clique}$ and $\mathit{rsa\text{-}sub}$ respectively, and they are compared to the BDD from Section 4.2 which uses both *gapfree*, *limited* and *upper bound*. Using these implementations, we conducted an experiment on all topologies in the Topology Zoo [51] to evaluate how the methods compare in different aspects.

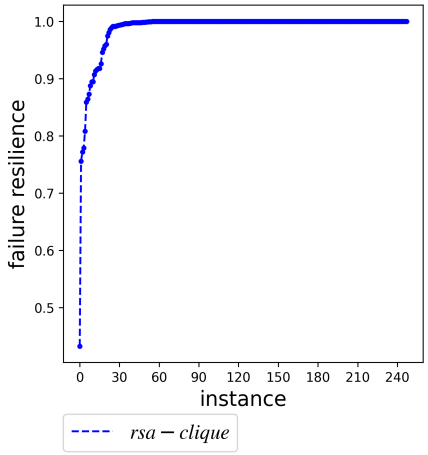
In Figure 7a, it can be seen that the maximum number of demands that can be solved for the RSA problem is significantly higher when using $\mathit{rsa\text{-}sub}$ compared to using $\mathit{rsa\text{-}clique}$ and the baseline rsa. However, as shown in Figure 7b, the resource usage for five demands is also significantly higher with $\mathit{rsa\text{-}sub}$, making it unsuitable for our main work due to the importance of resource efficiency. Conversely, using $\mathit{rsa\text{-}clique}$ does not lead to a significant increase in resource usage. Using $\mathit{rsa\text{-}clique}$ thus appears promising from the perspective of preserving optimality, but we also measured its performance in terms of k-link failure scenarios. Specifically, we evaluated how many scenarios were actually solved by the BDD out of of all 3-link failure scenarios where a solution is possible for six demands. As seen in Figure 7c, for most topologies, the solutions in $\mathit{rsa\text{-}clique}$ were fully 3-link failure resilient. However, for a notable number of instances, it was not 3-link failure resilient. Given the inability to generally guarantee k-link failure resilience, $\mathit{rsa\text{-}clique}$ is also not viable for use in our main work.



(a) Cactus plot of maximum number of demands solved.



(b) Cactus plot of usage for 5 demands.



(c) Cactus plot of percent failure resilience for 6 demands and 3 link failures.

Fig. 7: Results of improvements run on Topology Zoo.

References

[1] Cisco Systems, Inc. *Cisco Annual Internet Report (2018–2023) White Paper*. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. 2020.

[2] Gerd E Keiser. "A review of WDM technology and applications". In: *Optical Fiber Technology* 5.1 (1999), pp. 3–39.

[3] Dao Thanh Hai, Michel Morvan, and Philippe Gravey. "Combining heuristic and exact approaches for solving the routing and spectrum assignment problem". In: *IET Optoelectronics* 12.2 (2018), pp. 65–72. DOI: https://doi.org/10.1049/iet-opt.2017.0013. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-opt.2017.0013. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-opt.2017.0013.

[4] C. Gyorgyi et al. "SyRep: Efficient Synthesis and Repair of Fast Re-Route Forwarding Tables for Resilient Networks". In: *Proceedings of the 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'24)*. To appear. IEEE, 2024, pp. 1–12.

[5] C. Gyorgyi et al. "SyPer: Synthesis of Perfectly Resilient Local Fast Re-Routing Rules for Highly Dependable Networks". In: *IEEE International Conference on Computer Communications (INFOCOM'24)*. To appear. IEEE, 2024, pp. 1–10.

[6] Marco Chiesa et al. "A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks". eng. In: *IEEE Communications surveys and tutorials* 23.2 (2021), pp. 1253–1301. ISSN: 1553-877X.

[7] G Corfield. *British Airways' latest total inability to support upwardness of planes caused by Amadeus system outage*. 2018. URL: https://www.theregister.co.uk/2018/07/19/amadeus_british_airways_outage_load_sheet/.

[8] C Gibbs. *ATT's 911 outage result of mistakes made by ATT, FCC's Pai says*. 2017. URL: https://www.fiercewireless.com/wireless/at-t-s-911-outage-result-mistakes-made-by-at-t-fcc-s-pai-says.

[9] Dylan Tweney. *5-minute outage costs Google $545,000 in revenue*. 2013. URL: http://venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue.

[10] Metro Ethernet. *Technical specification MEF 2 requirements and framework for Ethernet service protection in metro Ethernet networks*. Tech. rep. Metro Ethernet, 2004.

[11] Alberto Castro et al. "On the benefits of multi-path recovery in flexgrid optical networks". In: *Photonic network communications* 28 (2014), pp. 251–263.

[12] KDR Assis et al. "Protection by diversity in elastic optical networks subject to single link failure". In: *Optical Fiber Technology* 75 (2023), p. 103208.

[13] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. "Understanding network failures in data centers: measurement, analysis, and implications". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. 2011, pp. 350–361.

[14] Gustav S. Bruhns et al. *ExpectAll source code*. https://github.com/Hebbe1234/ExpectAll [Accessed: 2024]. 2024.

[15] Rajiv Ramaswami and Kumar N Sivarajan. "Routing and wavelength assignment in all-optical networks". In: *IEEE/ACM Transactions on networking* 3.5 (1995), pp. 489–500.

[16] Biswanath Mukherjee. "Optical communication networks". In: *(No Title)* (1997).

[17] Hui Zang, Jason P Jue, Biswanath Mukherjee, et al. "A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks". In: *Optical networks magazine* 1.1 (2000), pp. 47–60.

[18] Imrich Chlamtac, Aura Ganz, and Gadi Karmi. "Lightpath communications: An approach to high bandwidth optical WAN's". In: *IEEE transactions on communications* 40.7 (1992), pp. 1171–1182.

[19] Konstantinos Christodoulopoulos, Ioannis Tomkos, and Emmanuel A Varvarigos. "Elastic bandwidth allocation in flexible OFDM-based optical networks". In: *Journal of Lightwave Technology* 29.9 (2011), pp. 1354–1366.

[20] Yang Wang, Xiaojun Cao, and Yi Pan. "A study of the routing and spectrum allocation in spectrum-sliced Elastic Optical Path networks". In: *2011 Proceedings IEEE INFOCOM*. 2011, pp. 1503–1511. DOI: 10.1109/INFCOM.2011.5934939.

[21] Yasutaka Miyagawa et al. "Bounds for two static optimization problems on routing and spectrum allocation of anycasting". In: *Optical switching and networking* 31 (2019), pp. 144–161.

[22] Junjia Zhang, Peng Miao, and Fuyong Zhang. "On optimal routing and spectrum allocation in elastic optical networks". In: *2023 2nd International Conference on Big Data, Information and Computer Network (BDICN)*. 2023, pp. 284–287. DOI: 10.1109/BDICN58493.2023.00066.

[23] Luis Velasco Esteban et al. "Modeling the Routing and Spectrum Allocation Problem for Flexgrid Optical Networks". In: *Photonic Network Communication* 24 (Jan. 2013), pp. 177–186. DOI: 10.1007/s11107-012-0378-7.

[24] Jiading Wang, Maiko Shigeno, and Qian Wu. "ILP models and improved methods for the problem of routing and spectrum allocation". In: *Optical Switching and Networking* 45 (2022), p. 100675. ISSN: 1573-4277. DOI: https://doi.org/10.1016/j.osn.2022.100675. URL: https://www.sciencedirect.com/science/article/pii/S157342772200011X.

[25] Miroslaw Klinkowski and Krzysztof Walkowiak. "Routing and Spectrum Assignment in Spectrum Sliced Elastic Optical Path Network". In: *IEEE Communications Letters* 15 (Aug. 2011), pp. 884–886. DOI: 10.1109/LCOMM.2011.060811.110281.

[26] Goran Markovic. "Routing and spectrum allocation in elastic optical networks using bee colony optimization". In: *Photonic Network Communications* 34 (Dec. 2017). DOI: 10.1007/s11107-017-0706-z.

[27] Dao Thanh Hai and Kha Manh Hoang. "An efficient genetic algorithm approach for solving routing and spectrum assignment problem". In: *2017 international conference on recent advances in signal processing, telecommunications & computing (SigTelCom)*. IEEE. 2017, pp. 187–192.

[28] Fernando Lezama et al. "Solving routing and spectrum allocation problems in flexgrid optical networks using pre-computing strategies". In: *Photonic Network Communications* 41 (Feb. 2021), pp. 1–19. DOI: 10.1007/s11107-020-00918-4.

[29] Takafumi Tanaka and Masayuki Shimoda. "Pre-and post-processing techniques for reinforcement-learning-based routing and spectrum assignment in elastic optical networks". In: *Journal of Optical Communications and Networking* 15.12 (2023), pp. 1019–1029.

[30] Liufei Xu et al. "Deep reinforcement learning-based routing and spectrum assignment of EONs by exploiting GCN and RNN for feature extraction". In: *Journal of Lightwave Technology* 40.15 (2022), pp. 4945–4955.

[31] Tao Gao et al. "Distributed sub-light-tree based multicast provisioning with shared protection in elastic optical datacenter networks". In: *Optical Switching and Networking* 31 (2019), pp. 39–51.

[32] Gangxiang Shen, Yue Wei, and Sanjay K Bose. "Optimal design for shared backup path protected elastic optical networks under single-link failure". In: *Journal of Optical Communications and Networking* 6.7 (2014), pp. 649–659.

[33] Narendra K. Singhal, Canhui Ou, and Biswanath Mukherjee. "Cross-sharing vs. self-sharing trees for protecting multicast sessions in mesh networks". In: *Computer Networks* 50.2 (2006). Optical Networks, pp. 200–206. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2005.05.024. URL: https://www.sciencedirect.com/science/article/pii/S1389128605001404.

[34] Pallavi Athe and Yatindra Singh. "Improved double cycle and link pair methods for two-link failure protection". In: *Telecommunication Systems* 74 (May 2020). DOI: 10.1007/s11235-019-00637-w.

[35] Xin Li et al. “Analysis and Modeling of k-regular and k-connected Protection Structure in Ultra-High Capacity Optical Networks”. In: *China Communications* 12 (Mar. 2015), pp. 106–119. DOI: 10.1109/CC.2015.7084369.

[36] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2024. URL: https://www.gurobi.com.

[37] Wikipedia. *List of cities in Germany by population*. URL: https://en.wikipedia.org/wiki/List_of_cities_in_Germany_by_population.

[38] Wikipedia. *Norden, Lower Saxony*. URL: https://en.wikipedia.org/wiki/Norden,_Lower_Saxony.

[39] Toshikazu Sakano et al. “A study on a photonic network model based on the regional characteristics of Japan”. In: *PN2013-1* 113.91 (2013), pp. 1–6.

[40] N.S. Johansen et al. “FBR: Dynamic Memory-Aware Fast Rerouting”. In: *IEEE Global Internet (GI) Symposium 2022*. IEEE, 2022, pp. 55–60. DOI: 10.1109/CloudNet55617.2022.9978819.

[41] Yusuke Hirota Kosuke Kubota Tosuke Tanigawa and Hideki Tod. “Crosstalk-aware Resource Allocation Based on Optical Path Adjacency and Crosstalk Budget for Space Division Multiplexing Elastic Optical Networks”. In: ().

[42] Kosuke Kubota et al. “Crosstalk-Aware Resource Allocation Based on Optical Path Adjacency and Crosstalk Budget for Space Division Multiplexing Elastic Optical Networks”. In: *IEICE Transactions on Communications* 107.1 (2024), pp. 27–38.

[43] C. Y. Lee. “Representation of switching circuits by binary-decision programs”. In: *The Bell System Technical Journal* 38.4 (1959), pp. 985–999. DOI: 10.1002/j.1538-7305.1959.tb01585.x.

[44] Akers. “Binary decision diagrams”. In: *IEEE Transactions on computers* 100.6 (1978), pp. 509–516.

[45] Randal E Bryant. “Graph-based algorithms for boolean function manipulation”. In: *Computers, IEEE Transactions on* 100.8 (1986), pp. 677–691.

[46] Randal Bryant. “Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams”. In: *ACM Computing Surveys* 24 (Mar. 2003). DOI: 10.1145/136035.136043.

[47] Kim Guldstrand Larsen et al. “AllSynth: Transiently Correct Network Update Synthesis Accounting for Operator Preferences”. In: *International Symposium on Theoretical Aspects of Software Engineering*. Springer. 2022, pp. 344–362.

[48] Soumya Eachempati et al. “Reconfigurable BDD based quantum circuits”. In: *2008 IEEE International Symposium on Nanoscale Architectures*. 2008, pp. 61–67. DOI: 10.1109/NANOARCH.2008.4585793.

[49] Andreas Rauchenecker and Robert Wille. “An efficient physical design of fully-testable BDD-based circuits”. In: *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. 2017, pp. 6–11. DOI: 10.1109/DDECS.2017.7934560.

[50] Henrik Reif Andersen. “An introduction to binary decision diagrams”. In: *Lecture notes, available online, IT University of Copenhagen* 5 (1997).

[51] Simon Knight et al. “The internet topology zoo”. In: *IEEE Journal on Selected Areas in Communications* 29.9 (2011), pp. 1765–1775.

[52] *Python package tulip-control/dd*. https://github.com/tulip-control/dd?tab=readme-ov-file [Accessed: 2024]. 2024.

[53] *Python package tulip-control/dd*. https://web.mit.edu/sage/export/tmp/y/usr/share/doc/polybori/cudd/cuddIntro.html [Accessed: 2024]. 2024.

[54] Gustav S. Bruhns et al. “AllRight: A BDD Based Approach to the Routing and Wavelength Assignment Problem”. In: *Aalborg University Student Projects* (2023). ID: alma9921650770905762.

[55] J MyıELsKı. “Sur le coloriage des graphes”. In: *Colloq. Math.* Vol. 3. 1955, p. l6.

Appendices

1 Demand Generation

The procedure for generating demands is described by Algorithm 1.

Algorithm 1 Demand generation algorithm

Input: Topology $G = (V, E, src, tgt)$, gravity function *gravity*, number of demands k, max demand size m.

Output: Set of demands D.

1: **procedure** GENERATEDEMANDS(G, *gravity*, k, m)
2: $D \leftarrow \emptyset$
3: $totalGravity = \sum_{v \in V} gravity(v)$
4: **for** i = 1 **to** k **do**
5: $size \leftarrow$ n, where n is randomly chosen in range $1 \leq n \leq m$
6: $source \leftarrow s \in V$, chosen by probability distribution $\frac{gravity(s)}{totalGravity}$
7: $target \leftarrow t \in (V \setminus source)$, chosen by probability distribution $\frac{gravity(t)}{totalGravity - gravity(source)}$
8: $D \leftarrow D \cup \{(source, target, size)\}$
9: **end for**
10: **return** D
11: **end procedure**

As input, the algorithm takes a network topology, a gravity function that maps a node from the network topology $G = (V, E, src, tgt)$ to the population size of the city represented by the node, the number of demands to generate k, and the maximum size m of any demand. It then outputs a set of demands D where max size of any demand $d \in D$ is m and the source and target node of each demand is chosen based on the gravity function. On lines 2-3, the set of demands is initialized as an empty set, and the integer variable *totalGravity* is initialized to the sum of all population sizes for the nodes. Lines 4-8 iteratively add one demand to the set of demands D until k demands have been generated. For iteration $1 \leq i \leq k$, we uniformly choose a size $1 \leq n \leq m$ for demand d_i on line 5. Then, on line 6-7, we pick a source $s \in V$ and target $t \in V$, $s \neq t$, with respect to a probability distribution that is based on the mapping *gravity* and the population total *gravityTotal*. Specifically, nodes with large population sizes have a greater chance of being picked as a source or target node than nodes with smaller population sizes. Finally, on line 8, we add demand d_i with source s, target t and size n to the set of demands D. After the k iterations, we output the set of demands D.

2 Limited counter example

Figure 8 shows a RSA problem, and the candidate channels for each demand, when *limited* is applied. The problem is solvable without *limited* because when d_0 and d_1 uses the first four and last four slots respectively, there are 12 consecutive slots available on all the other edges that they share with the other demands, allowing them to utilize the spectrum completely. Using the channels generated by *limited*, d_1 can only use up to slot 12, which means that there are not 12 consecutive slots available on the edges with other demands. Due to how the other demands are constructed with lowest possible product being 12, we can only solve the problem if there are 12 consecutive slots available for them all.

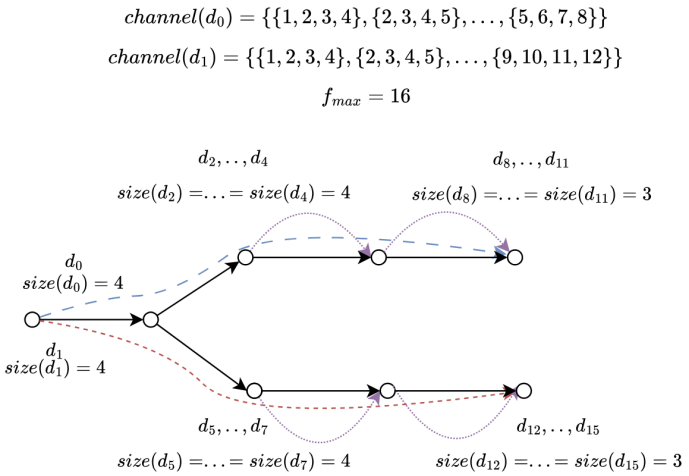


Fig. 8: Counter example of limited using 16 demands, with the demand ordering $d_0, d_1, ...d_{15}$.

3 Lightpath-preserving solution tables

Tables showing the percentage of times that the BDDs could find a lightpath-preserving failover solution as well as the percentage of overall feasible lightpath-preserving solutions for 1000 queries, 1-9 demands, and 1-5 link failures. For instance, on DT for 9 demands, 5 link failures, the BDD found the lightpath-preserving solution in 63% of the failure scenarios.

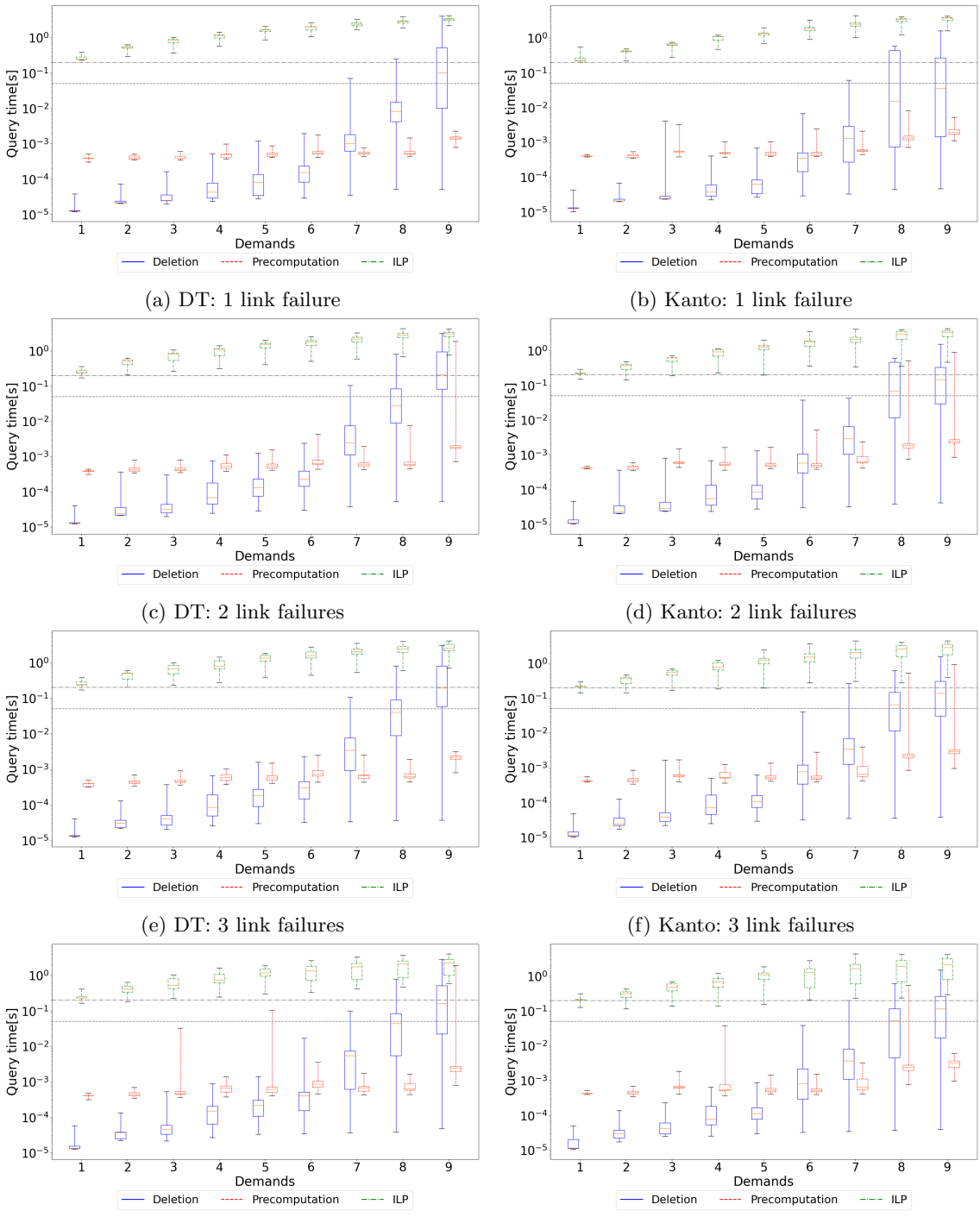
Table 3: DT

BDD% / Possible%		Failures				
		1	2	3	4	5
Demand	1	100 / 100	100 / 100	100 / 100	99 / 99	98 / 98
	2	100 / 100	99 / 99	96 / 96	94 / 94	89 / 89
	3	100 / 100	98 / 98	94 / 94	91 / 91	84 / 84
	4	100 / 100	98 / 98	93 / 93	87 / 87	78 / 78
	5	100 / 100	96 / 96	89 / 89	83 / 83	72 / 72
	6	91 / 100	82 / 95	69 / 84	57 / 76	48 / 63
	7	91 / 100	82 / 95	67 / 82	52 / 71	42 / 56
	8	82 / 100	68 / 95	54 / 82	40 / 71	31 / 56
	9	86 / 100	72 / 94	55 / 80	43 / 69	34 / 54

Table 4: Kanto

BDD% / Possible%		Failovers				
		1	2	3	4	5
Demand	1	100 / 100	100 / 100	99 / 99	98 / 98	96 / 96
	2	100 / 100	99 / 99	96 / 96	92 / 92	89 / 89
	3	100 / 100	99 / 99	96 / 96	91 / 91	87 / 87
	4	100 / 100	96 / 96	91 / 91	80 / 80	74 / 74
	5	92 / 100	82 / 96	71 / 88	61 / 78	49 / 71
	6	92 / 100	79 / 93	65 / 82	51 / 68	37 / 56
	7	84 / 100	67 / 93	54 / 82	40 / 67	28 / 56
	8	86 / 100	71 / 93	58 / 82	44 / 67	35 / 56
	9	87 / 100	69 / 93	53 / 82	39 / 67	26 / 56

4 Optimal Query time graphs



(a) DT: 1 link failure

(b) Kanto: 1 link failure

(c) DT: 2 link failures

(d) Kanto: 2 link failures

(e) DT: 3 link failures

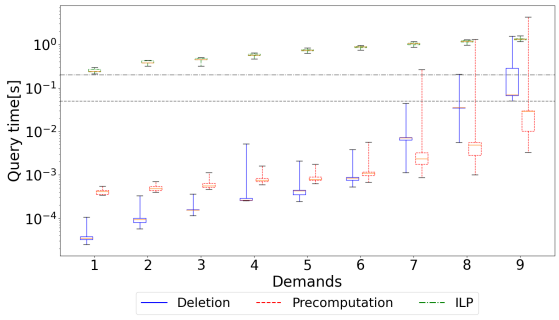
(f) Kanto: 3 link failures

(g) DT: 4 link failures

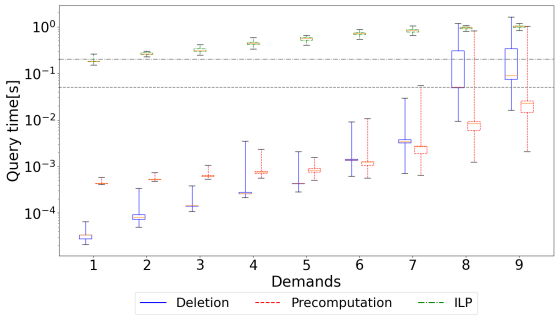
(h) Kanto: 4 link failures

Fig. 9: Boxplots of query time to find an optimal solution for 1000 queries on DT and Kanto 11.

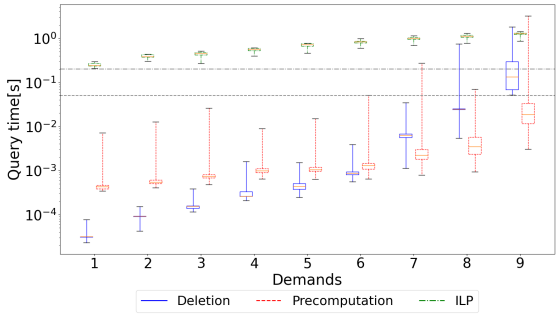
5 Non-changing Query time graphs



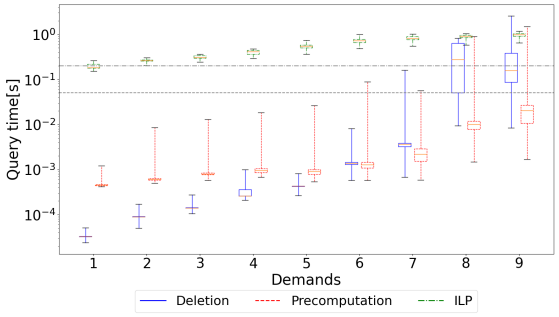
(a) DT: 1 link failure



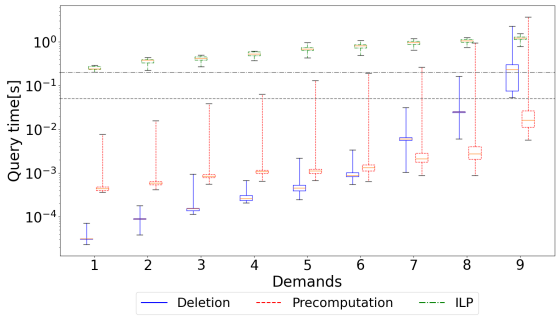
(b) Kanto: 1 link failure



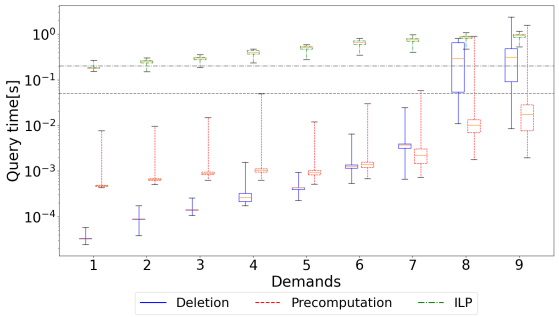
(c) DT: 2 link failures



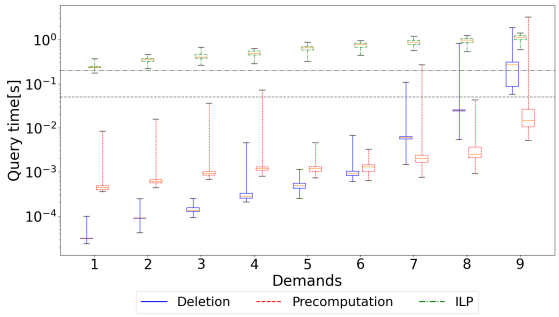
(d) Kanto: 2 link failures



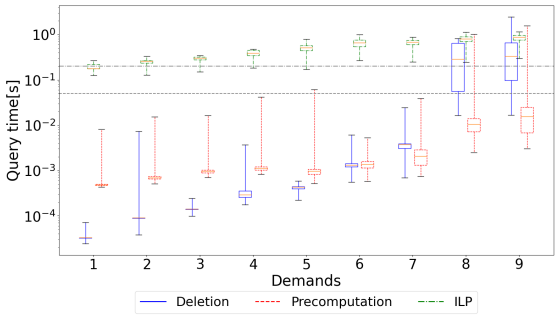
(e) DT: 3 link failures



(f) Kanto: 3 link failures



(g) DT: 4 link failures



(h) Kanto: 4 link failures

Fig. 10: Boxplots of total query time to find an optimal lightpath-preserving solution for 1000 queries on DT and Kanto 11.