Impairment-Aware Path Selection in Translucent Optical Networks

Fernando A. Kuipers¹, Anteneh A. Beshir¹, Arial Orda², and Piet F.A. Van Mieghem¹

¹Network Architectures and Services
Delft University of Technology
Mekelweg 4, 2628 CD Delft
The Netherlands
Email: {F.A.Kuipers, A.A.Beshir, P.F.A.VanMieghem}@tudelft.nl

²Department of Electrical Engineering, Technion
Israel Institute of Technology
Haifa, Israel 32000
Email: ariel@ee.technion.ac.il

August 15, 2009

Abstract

Physical impairments, such as noise and signal distortions, negatively affect the quality of information transport in optical networks. The effect of physical impairments augments with distance and bit rate of the signal to the point that it becomes detrimental to the information transfer. To reverse the effect of physical impairments, the signal needs to be regenerated at nodes that have regeneration capabilities. Regenerators are costly and therefore usually only sparsely placed in the network, which in that case is referred to as a translucent network. This paper deals with impairment-aware routing in translucent networks. The problem is to allocate a lightpath between a source and a destination, such that the segments of the lightpath between two regeneration nodes have an acceptable level of physical impairment. We study the complexity of impairment-aware path selection and show that there is an efficient exact solution if routing loops are allowed; however, when paths need to be simple, we establish that efficient approximation algorithms are unlikely to exist. We subsequently propose exact and heuristic algorithms for impairment-aware path selection and through simulations show that our heuristic TIARA is computationally efficient and performs very close to our exact algorithm EIARA.
1 Introduction

In transparent all-optical networks, the signal is transmitted in the optical domain from the source to the destination node, without any conversion to the electrical domain. If the signal is not regenerated at intermediate nodes, noise and signal distortions are accumulated along the physical path. The noise and signal distortions are known as physical impairments and degrade the quality of the received signal. Especially for long distances and high bit rates, the signal degradation may lead to an unacceptable bit error rate (BER). Physical impairments can be classified into two categories: linear and non-linear impairments. Linear impairments are independent of signal power and affect wavelengths individually. Non-linear impairments generate not only dispersion on each channel, but also crosstalk between channels.

In order to overcome physical impairments, re-amplification, re-shaping, and re-timing, which are collectively known as 3R regeneration, are used at intermediate nodes. Even though optical 3R regenerations have been demonstrated in laboratories, only electrical 3R regenerations are currently reliable and economically viable [3]. In the latter case, signal regeneration is achieved through optical to electrical and then back to optical (O-E-O) conversions. Since regenerators are costly, it is practical to have sparse regeneration capacity in the network. A network that uses sparse regeneration is known as a translucent network. The results presented in this paper hold for both optical and electrical 3R regenerations.

In this paper, we deal with impairment-aware path selection. In the literature, different types of cost functions have been suggested for links and nodes to represent their physical impairments during the path-selection process. These include the distance of a link [20], a cost which is a function of the residual dispersion parameter [5], the Four-Wave Mixing (FWM) crosstalk [15], the signal quality Q-factor [20], and the noise variance [10]. These different impairment cost functions will have a corresponding impairment threshold that relates to a certain level of signal quality. Our work is independent of the impairment cost function used and is applicable to any additive link or node cost function.

The outline of this paper is as follows. Related work is presented in Section 2. In Section 3, we give a formal definition of the impairment-aware routing problem. We study the complexity of the problem and provide exact and heuristic algorithms under two scenarios: when loops are allowed (Section 4) and when only loopless routing is permitted (Section 5). Simulation results for our algorithms are given in Section 6. Finally, conclusions are presented in Section 7.

2 Related Work

Numerous routing and wavelength assignment (RWA) algorithms have been proposed in the literature for WDM optical networks. A majority of these algorithms assume ideal physical-layer conditions. A review of some of these
algorithms is given by Zang et al. [19]. Recently, different approaches have been considered to provide physical impairment-aware RWA algorithms.

In one approach, the path and the wavelength of a lightpath are computed in the traditional way without taking into account the physical impairments, and subsequently the quality of the selected lightpath is tested against physical-layer impairments [2], [16]. New paths are computed if the candidate paths do not meet the physical impairments.

In another approach, the physical-layer impairment values are considered in the routing and/or wavelength assignment decisions [7], [10], [14]. In these works, the physical-layer information is used as weight of the links in order to compute the minimum-cost lightpath, after which compliance with the impairment threshold is verified. A detailed survey of impairment-aware RWA algorithms is given by Azodolmolky et al. [3].

The afore-mentioned approaches do not consider the physical impairment values and threshold between regenerator nodes in the path-selection process. Instead, they are verified only after the path is computed. In this paper, we study the impairment-aware path-selection part of the routing and wavelength assignment problem. We also provide a detailed study into the complexity of the problem and propose both exact and heuristic algorithms. Our algorithms can be used for both linear and non-linear impairments, as long as they are provided as link weights. These link weights can, for example, represent worst-case physical impairment values of the respective links.

3 Network Model

Fig. 1 shows some of the components that make up a typical transmission system of a translucent optical transport network (OTN). In this model, a node is mainly composed of an all-optical switch, optional regenerators, transponders, multiplexers, demultiplexers, and pre- and post-amplifiers; whereas a fiber link is a WDM line system comprising of fibers and amplifiers. At a given node, transponders modulate electrical signals onto distinct wavelengths. These wavelengths are then multiplexed and the multiplexed signal is pre-amplified before being propagated through the WDM line. Finally, at the receiver the signal is post-amplified and de-multiplexed into individual signals. A link has a single fiber in each direction, each containing a number of wavelengths to be used by lightpaths. In Fig. 1, only one direction is shown. We assume that there is no conversion at any of the nodes, which implies that a lightpath should use the same wavelength on all of its links.

At each node, there are add and drop ports for data to locally enter and leave the network. Each incoming signal is demultiplexed and switched inside a translucent node using an all-optical switching fabric, which can switch an optical signal from any input port to any other output port. In a translucent network, an optional pool of regenerators can be attached between some transmitters and receivers in order to facilitate regeneration as shown in Fig. 1. We assume that the optical switches at the translucent nodes have enough ports...
to support incoming signals as well as regenerated signals. It is possible that some nodes do not have regenerators, thus providing only the service of locally adding and dropping of wavelengths.

Figure 1: Network components that make up a translucent WDM OTN.

We define a regeneration segment of a lightpath to be a transparent segment (i.e., one or more links) between two regenerator nodes (including source and destination nodes) of the lightpath. A lightpath can be made up of multiple regeneration segments. There is no need for a lightpath to be regenerated at the source and destination nodes. A transparent lightpath is a special regeneration segment that is not regenerated anywhere along its path. After a transit signal is regenerated, its original physical features are restored. From a physical impairment point of view, the effect of physical impairments along the path followed to reach the regenerator node is completely removed.

Based on the described network model, we define the impairment-aware routing problem as follows:

**Problem 1** The impairment-aware routing problem: The physical optical network is modeled as a graph $G(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of $N$ nodes and $\mathcal{L}$ is the set of $L$ links. Associated with each fiber link $(u, v) \in \mathcal{L}$ is a physical impairment $r(u, v)$. $\mathcal{N}_R \subseteq \mathcal{N}$ represents the set of $R$ nodes that have (spare) regeneration capacity. A request is represented by the tuple $(s, d, \Delta)$, where $s, d \in \mathcal{N}$ are the source and destination nodes of the request and $\Delta$ represents the threshold for the physical impairment. The impairment-aware routing problem is to find a route from source to destination that does not exceed the threshold $\Delta$ on any of its regeneration segments.

We illustrate this problem using the example network in Fig. 2(a) for a request $(s, d, 5)$. In this example, the shortest path from $s$ to $d$ goes via the direct link $(s, d)$, but this path violates the impairment threshold, i.e., $r(s, d) = 6 > \Delta$. 
The only feasible path is $s - t - d$, where $t$ is a regenerator node, because for the regeneration segments holds $r(s, t) = r(t, d) = 5 \leq \Delta$.

Figure 2: Example networks with request $(s, d, 5)$. (a) There is a feasible path through regeneration node $t$. (b) There is no feasible simple path.

Consider now the instance in Fig. 2(b), where, given a request $(s, d, 5)$, there is a feasible walk $s - 2 - t - 2 - d$, but there is no feasible simple path for the impairment-aware routing problem. It is of interest to consider a variant of the impairment-aware routing problem, in which only simple paths are admitted as solutions; indeed, such restrictions may be due to scarcity of resources (link or node capacity) or management considerations. In the following two sections, we shall consider both variants, i.e., impairment-aware routing with and without loops.

4 Routing with loops

We first present a Polynomial-time Impairment-Aware Routing Algorithm (PIARA) for finding a path from source $s$ to destination $d$ subject to an impairment threshold $\Delta$ and for the case that nodes and links may be revisited.

Algorithm 1 PIARA($G, s, d, N_R, \Delta$)

1. Find the shortest paths $\{P^*_i \rightarrow j\}$ between all nodes $i, j \in N_R \cup \{s, d\}$, for which $r(P^*_i \rightarrow j) \leq \Delta$.

2. Make a graph $G'$ consisting of nodes $N_R \cup \{s, d\}$ and links $(i, j)$ (e.g., with weight $r(P^*_i \rightarrow j)$) if the shortest path $P^*_i \rightarrow j$ was feasible in the original graph $G$.

3. Find a (shortest) path from $s$ to $d$ in $G'$ and substitute the links of the path in $G'$ with the corresponding subpaths $P^*_i \rightarrow j$ in $G$.

Figure 3 gives an example of the execution of PIARA when $\Delta = 5$. 
Figure 3: (a) The graph $G$ with source $s$, destination $d$, and regenerator nodes $i$ and $j$. Of the shortest paths between $s, i, j$, and $d$, only the shortest path between $i$ and $d$ exceeds $\Delta$. (b) The graph $G'$ in which the shortest path is $s - m - d$, which maps to the solution $s - j - m - d$ as returned by PIARA.

The algorithm PIARA assumes that link or node capacities are not confining, even when traversed multiple times. When link or node capacities are confining we may need to find loop-free paths, which is considered in Section 5. Instead of only using one path, our objective may be to push as much flow from a source to a destination, possibly using multiple paths that may have loops and obeying impairment and capacity constraints. This problem is referred to as the impairment-aware maximum flow problem, which is considered in the next section.

4.1 Max-Flow with loops

Problem 2 The impairment-aware maximum flow problem: Consider a physical optical network $G(N, L)$. Associated with each fiber link $(u, v) \in L$ are a physical impairment $r(u, v)$ and a capacity $\mu(u, v)$. $N_R \subseteq N$ represents the set of $R$ nodes that have (spare) regeneration capacity. A request is represented by the tuple $(s, d, \Delta)$, where $s, d \in N$ are the source and destination nodes of the request and $\Delta$ represents the impairment threshold. The impairment-aware maximum flow problem is to send as much flow as possible from $s$ to $d$, where the (possibly non-simple) paths used to transport (fractions of) the flow do not exceed the threshold $\Delta$ on any of their regeneration segments nor exceed the link capacities.
Theorem 1  The impairment-aware maximum flow problem is NP-hard.

Figure 4: Example to illustrate that computing an impairment-aware maximum flow is NP-hard.

To prove our theorem we make use of the NP-hard partition problem [9], which is defined as follows.

Problem 3  The partition problem: Give a set of weights \( a_i \in A \), \( a_i \geq 0 \) for \( i = 1, \ldots, n \), where \( S = \sum_{i=1}^{n} a_i \). Find a subset \( I \subseteq A \) such that \( \sum_{a_i \in I} a_i = \sum_{a_i \in A \setminus I} a_i = \frac{S}{2} \).

Proof.  We shall only provide a proof for undirected graphs. The directed case follows analogously. Consider the graph in Figure 4, where all vertical links have a capacity of two units and all remaining links have unit capacity. For the weights associated with the labeled links \( a_i \in A \), \( a_i \geq 0 \), holds that \( S = \sum_{i=1}^{n} a_i \).

All links without labels have zero weight. We choose \( \Delta = S + 1 \), which means that any feasible flow must pass through the regenerator. A maximum flow can be split over multiple paths and any fractional flow must be coupled to a path to ascertain whether it is feasible or not. Furthermore, since the source is connected to a single link of unit capacity, we know that the maximum flow cannot exceed one. Our proof consists of showing that we can only establish a maximum flow of 1 if we solve the partition problem.

Since we ship one unit of flow from \( s \) to \( t \) and back from \( t \) to \( d \), the accumulated flow through two links precisely above each other must sum to 2. If a portion \( f \) is flowing through a top link from \( s \) to \( t \), then the remaining portion \( 1-f \) must be flowing from \( s \) to \( t \) on the corresponding bottom link. Since the horizontal links have unit capacity, this means we can only direct a portion \( 1-f \) on that top link and a portion \( f \) on that bottom link when going from \( t \) to \( d \). The more flow is sent through the top links labeled \( a_i \) from \( s \) to \( t \), the more capacity is available on the corresponding zero-weight links below. Consequently, flows on feasible paths from \( s \) to \( t \) that have a length \( < \Delta \) will result in some flows having to return from \( t \) to \( d \) on paths of length \( > \Delta \).

If flows are split (not necessarily evenly) over \( k \) feasible paths from \( s \) to \( t \), we can always delete one of these \( k \) paths and add its flow portion to another path with length larger or equal, without violating the constraint (since the larger path satisfies the constraint and because the sum of the flows from \( s \) to \( t \) through a labeled link and its corresponding zero-weight link add up to 1).

And since we push one unit of flow (the maximum possible flow value) from \( s \)
to \(d\), when we shift a portion of flow from a top link to a bottom link (or vice versa), we can also shift a similar portion flowing in the opposite direction from the bottom link to the top link (or vice versa). As these shifts correspond to feasible paths from \(s\) to \(t\), we will not violate any constraint. Moreover, we will be shifting to smaller length paths on the segment from \(t\) to \(d\). By iterating this process we will end up with one path from \(s\) to \(t\). If there is only one path from \(s\) to \(t\) (sending one unit of flow), then there is also only one remaining path that can be used from \(t\) to \(d\). However, if this path from \(s\) to \(t\) has length \(<\Delta\), this means that the corresponding path from \(t\) to \(d\) must have length \(\geq \Delta\). A feasible max-flow solution therefore only exists if the length of the used path segments to and from the regenerator is precisely \(\Delta\), which requires solving partition.

We present a pseudo-polynomial-time LP formulation to solve the maximum impairment-aware multicommodity flow problem. We refer to this LP formulation as the Pseudo-polynomial Impairment-aware Multicommodity Flow (PIMF) algorithm. The maximum multicommodity flow problem is a generalization of the maximum flow problem, where multiple source-destination pairs are considered simultaneously instead of only one pair. We make the following assumptions: at any regeneration node, there is no limit on the number of available regenerators, and a flow passing through a regeneration node will be regenerated. In addition, we assume, w.l.o.g., that the link weights (i.e., the physical impairment values) are positive integers. We provide the LP formulation for directed networks (it can be extended for undirected networks by adding equations that specify that the flow is in both directions).

The LP formulation depends on \(\Delta\) and thus has a pseudo-polynomial time complexity. Pseudo-polynomial complexity indicates that the impairment-aware maximum flow problem is only weakly NP-hard and that, by properly rounding and scaling, an efficient approximation algorithm can be derived. An approximation in this context refers to a solution for which the regeneration segments are bounded by \((1 + \epsilon)\Delta\) and whose time complexity is fully polynomial in the input and \(\frac{1}{\epsilon}\). We provide the outline of such an Impairment-aware Multicommodity Flow Approximation (IMFA) below.

**Theorem 2** Algorithm IMFA is a fully-polynomial approximation scheme for the maximum impairment-aware multicommodity flow problem.

**Proof.** For each link \((u, v) \in E\) we have that \(\frac{r\, (u, v) \, N \, \Delta}{\epsilon \, \Delta} \leq r' (u, v) \leq \frac{r\, (u, v) \, N \, \Delta}{\epsilon \, \Delta} + 1\) or equivalently \(r (u, v) \leq \frac{r' (u, v) \, N \, \Delta}{\epsilon \, \Delta} \leq r (u, v) + \frac{\Delta}{\epsilon}\). For any regeneration segment \(S\) follows

\[
r (S) = \sum_{(u,v) \in S} r\, (u,v) \leq \frac{\epsilon \, \Delta}{N} \sum_{(u,v) \in S} r' (u, v)
\]

Since \(\sum_{(u,v) \in S} r' (u, v) \leq \Delta' \leq \frac{N}{\epsilon} + N\) only regeneration segments \(S\) can (and will) be considered, for which holds

\[
r (S) \leq \frac{\epsilon \, \Delta}{N} \left( \frac{N}{\epsilon} + N \right) = (1 + \epsilon) \Delta
\]
Algorithm 2 PIMF

Indices:
- $i$: The commodity between nodes $s_i$ and $d_i$.
- $\mathcal{N}^-(u)$: Set of adjacent nodes of node $u$ whose links enter $u$.
- $\mathcal{N}^+(u)$: Set of adjacent nodes of node $u$ whose links exit $u$.

Constants:
- $\mu_{u,v}$: The link capacity of link $(u,v) \in \mathcal{L}$.
- $r_{u,v}$: The link weight of link $(u,v) \in \mathcal{L}$.

Variables:
- $x_{i,u,v}^r$: The amount of flow of commodity $i$ (from $s_i$ to $d_i$) on link $(u,v) \in \mathcal{L}$ whose distance from the last regenerator node (or the source node) is $r$.

Objective:
Maximize the total amount of flow:
\[
\text{Maximize: } \sum_i \sum_{u \in \mathcal{N}^+(s_i)} x_{i,u,v}^r
\]

Constraints:
Flow conservation constraints for non-regenerator nodes:
\[
\sum_{u \in \mathcal{N}^-(v)} x_{i,u,v}^r = \sum_{w \in \mathcal{N}^+(v)} x_{i,v,w}^{r+r_{v,w}}
\forall i; \forall v \in \mathcal{N} \backslash (\mathcal{N}_R \cup \{s_i, d_i\}); \forall r
\]

Flow conservation constraints for regenerator nodes:
\[
\sum_r \sum_{u \in \mathcal{N}^-(v)} x_{i,u,v}^r = \sum_{w \in \mathcal{N}^+(v)} x_{i,v,w}^{r-r_{v,w}}
\forall i; \forall v \in \mathcal{N}_R
\]

Impairment and capacity constraints
\[
x_{i,u,v}^r = 0 \quad \forall i; \forall u, v \in \mathcal{N}; \forall r > \Delta
\]
\[
\sum_r \sum_i x_{i,u,v}^r \leq \mu_{u,v} \quad \forall (u, v) \in \mathcal{L}
\]

Avoiding cycles at the source nodes
\[
\sum_{u \in \mathcal{N}^-(s_i)} x_{i,u,s_i}^r = 0 \quad \forall i; \forall r
\]
Algorithm 3 IMFA

1. For each link \((u, v) \in \mathcal{L}\) define \(r'(u, v) = \left\lfloor \frac{r(u, v)N}{\epsilon \Delta} \right\rfloor + 1\).

2. Define \(\Delta' = \left\lfloor \frac{N}{\epsilon} \right\rfloor + N\).

3. Run PIMF with \(r'(u, v)\) instead of \(r(u, v)\) and \(\Delta'\) instead of \(\Delta\).

The complexity of the algorithm is determined by \(\Delta' \leq (1 + \frac{1}{\epsilon}) N\) and the size of the input, and is therefore fully polynomial. ■

5 Routing Without Loops

In this section, we require the routing to be loop free.

Theorem 3 The impairment-aware loopless routing problem is strongly NP-complete\(^1\).

To prove that the problem is NP-hard, we shall use the Maximum Length-Bounded Disjoint Paths (MLBDP) problem [9], which is defined as follows.

Problem 4 The maximum length-bounded disjoint paths problem: Given an undirected graph \(G\), source \(s\) and destination \(d\), and positive integers \(b\) and \(K\), does \(G\) contain \(K\) or more mutually node-disjoint paths from \(s\) to \(d\), none involving more than \(b\) links?

\(^1\)“Strongly NP-complete” indicates that the problem remains NP-complete even if the link weights are bounded by a polynomial in the length of the input. Contrary to weakly NP-complete problems, these problems do not admit pseudo-polynomial time solutions.
The MLBDP problem was proven to be NP-complete for $b \geq 5$ by Itai et al. in [12] and later proven to be APX-hard$^2$ for $b \geq 5$ by Bley in [4].

**Proof.** When we are given a path it is easy to verify whether it obeys the threshold $\Delta$ or not. The problem is therefore in NP. We shall provide a reduction to the MLBDP problem to prove strong NP-completeness.

Any instance of the MLBDP problem can be transformed in polynomial time to an impairment-aware routing instance as follows. The source is split into $K$ sources $s_1, ..., s_K$ and the destination is split into $K$ destinations $d_1, ..., d_K$. Each of these sources (destinations) is connected to the same nodes as the original source (destination) node. So far all links have weight 1. We add a new source and connect it to $s_1$ with a link of weight $x - b$. For each pair of source nodes $(s_{2i}, s_{2i+1})$, for $i = 1, ..., \lfloor \frac{K-1}{2} \rfloor$, we add a new regenerator node and link it to $s_{2i}$ with weight $2ib$ and to $s_{2i+1}$ with weight $x - (2i+1)b$. For each pair of destination nodes $(d_{2i-1}, d_{2i})$, for $i = 1, ..., \lfloor \frac{K-1}{2} \rfloor$, we add a new regenerator node and link it to $d_{2i-1}$ with weight $(2i-1)b$ and to $d_{2i}$ with weight $x - 2ib$. The last node (either $s_K$ or $d_K$) is connected to a new destination node through a link with weight $Kb$. Fig. 5 visualizes this construction for $K = 4$. If we choose $\Delta = x + b$ and $x > 2Kb$, then solving the impairment-aware routing problem in the new graph provides a solution to the MLBDP problem. Moreover, since $Kb \leq 2(N - 1)$, we have that $\Delta = O(N)$, which on its turn means that the impairment-aware routing problem is strongly NP-complete. 

An algorithm is said to be an $\epsilon$-approximation for the impairment-aware routing problem if it returns a path whose regeneration segments have length (in terms of physical impairments) at most $(1 + \epsilon)\Delta$, with approximation factor $\epsilon > 0$. In practice, approximating on the threshold is not desirable, but since the feasibility problem has been proved strongly NP-complete, any other type of polynomial-time approximation (e.g., approximating on the total length while obeying the impairment threshold) cannot be found, unless P=NP. Moreover, as a corollary of the proof that the feasibility problem itself is strongly NP-complete, there does not exist a fully polynomial approximation scheme unless P=NP. Approximation schemes are consequently very costly (exponential in $\frac{1}{\epsilon}$) or violate $\Delta$ too much, rendering them less useful in practice.

**Theorem 4** In directed graphs, the impairment-aware routing problem cannot be approximated within a factor of $R + 1$, unless P=NP.

**Proof.** Consider Fig. 6 with $R = 1$, where we intend to find two node-disjoint paths for $(s_1, d_1)$ and $(s_2, d_2)$ inside the network cloud. Finding two node-disjoint paths in a directed network is an NP-hard problem [9]. We add three nodes $(s, d$ and a regenerator node $t)$ as shown in Fig. 6 and let all links have zero weight and set $\Delta = x$. Assume that there is a polynomial-time $\alpha$-approximation algorithm, with $\alpha < R+1 = 2$, for the impairment-aware routing problem. By solving the impairment-aware routing problem from node $s$ to $d$

$^2$APX-hard problems can be approximated within some constant factor, but not every constant factor (as with polynomial-time approximation schemes (PTAS)), unless P=NP.
through node $t$ using such algorithm, we can find two-node disjoint paths for $(s_1, d_1)$ and $(s_2, d_2)$, which contradicts that that problem is NP-hard in directed networks. We have demonstrated that we cannot approximate within a factor of 2. The extension to $R + 1$ follows analogously by considering $R$ network clouds where the link from $d_2$ goes to $s_1$ in another network cloud, except for the last network cloud, where the link goes from $d_2$ to the destination $d$.

Figure 6: Impairment-aware routing cannot be approximated within a factor of 2 in a directed network. The cloud represents a directed network with links of cost 0.

**Theorem 5** For any number of regenerators $R$, a shortest path algorithm that disregards whether a node is a regenerator node or not is an $(R+1)$-approximation algorithm for the impairment-aware routing problem.

**Proof.** Let $R = 1$. Given a network with a single regenerator node, compute the shortest path $P_{s \rightarrow d}^*$ from source to destination. The shortest path could include the regenerator node. If $r(P_{s \rightarrow d}^*) \leq \Delta$ we have found an optimal solution. Assume that $P_{s \rightarrow d}^*$ is not feasible and that the optimal solution consists of a subpath $P_{s \rightarrow t}^{opt}$ from the source to the regenerator node $t$ and a subpath $P_{t \rightarrow d}^{opt}$ from the regenerator node to the destination. Since $r(P_{s \rightarrow d}^*) \leq r(P_{s \rightarrow t}^{opt}) + r(P_{t \rightarrow d}^{opt}) \leq 2\Delta$, this proves our theorem. If $r(P_{s \rightarrow d}^*) > 2\Delta$, then no solution exists. The proof can be extended for any $R > 1$, since the subpaths of the optimal solution are always bounded by $\Delta$, leading to a length of $(R + 1)\Delta$, while the shortest path may approach this length from below without having any regenerators on its path.

Since good approximation schemes are unlikely to exist (as justified by Theorem 3), in the following sections we focus on exact and heuristic solutions.

### 5.1 Problem variants

In impairment-aware routing, the use of regeneration capabilities of a regenerator node is optional. Depending on how regenerator nodes are used, associating an objective with solving the impairment-aware routing problem can lead to several problem variants.
**Variant 1:** Find the shortest (in terms of physical impairment) feasible path. Regenerators can be used at no extra cost.

**Variant 2:** Given that each used regenerator has a cost of usage that will be added to the total path length, find the shortest feasible path.

**Variant 3:** Find a feasible path that uses the fewest number of regenerators. In case of a tie, the one with shortest length is returned.

Problem variants 2 and 3 can be transformed into problem variant 1 by splitting each regenerator node in the input graph $G$ into four nodes as shown in Fig. 7(a) for undirected networks and Fig. 7(b) for directed networks. In these figures, the link weight $x$ equals the cost of using the given regenerator in problem variant 2, while $x = \Delta$ in problem variant 3. We will focus on solving problem variant 1.

![Figure 7: Regenerator node splitting in (a) an undirected network, and (b) a directed network.](image)

In solving the impairment-aware shortest path routing problem, we have to take into account two parameters in the search process:

1. The total length $r(P)$ of a (sub)path $P$ accumulated since the source node.
2. The length $r_1(P)$ since the last used regenerator node (or the source node) along a (sub)path $P$.

The fact that $r_1(P)$ does not reflect an end-to-end property prevents a simple adoption of multi-parameter algorithms like SAMCRA [18]. Two search-space reducing techniques that are used in SAMCRA are the concept of non-dominance (or Pareto optimality) and the concept of look-ahead. We will demonstrate that, while the concept of non-dominance cannot be used, we can apply the look-ahead concept with some modifications.

### 5.1.1 Non-dominance

When solving multi-constrained routing problems, at any intermediate node, it does not make sense to consider a (sub)path that has worse weights (i.e., higher or equal in every metric) than another (sub)path. Such paths are said to be dominated and are discarded, thereby reducing the search-space. This non-dominance technique fails in impairment-aware routing as shown in Fig. 8. In this example, the request is $(s,d,9)$. At node 3, the subpath $P_1 = s - 3$ with $r(P_1) = r_1(P_1) = 8$ is dominated by the subpath $P_2 = s - 1 - 2 - 3$ with...
\[ r(P_2) = r_1(P_2) = 7. \] However, \( P_1 \) cannot be discarded since it is part of the only feasible path \( s - 3 - t - 2 - 1 - d. \)

Assuming non-negative link weights, the non-dominance principle prevents loops along a path. In its absence, we will have to check for loops explicitly.

![Graph](image.png)

**Figure 8:** An example wherein the concept of non-dominance fails for impairment-aware routing. The request is \((s, d, 9)\) and node \(t\) is a regenerator.

### 5.1.2 Look-ahead

Look-ahead refers to finding lower bounds on the weights of the remaining sub-path towards the destination in order to predict whether the current subpath will exceed any of the constraints. For multi-constrained routing, this information is built by computing, for each metric, the shortest paths tree rooted at the destination node to each node in the network. For the impairment-aware routing problem, we employ two look-ahead values for each node, i.e., the distance of the node to its nearest regeneration node and the shortest distance of the node to the destination node. The former is used to calculate whether the current segment of the given subpath will lead to a distance higher than \( \Delta \), while the latter is used to assess whether the lower bound on the end-to-end distance of the given subpath exceeds \((R + 1)\Delta\), since any feasible path can use a maximum of \( R \) regenerators.

### 5.2 Pruning the Graph

In a given graph, it may be necessary to check all paths between the source and destination nodes before concluding that a feasible path does not exist. This, in the worst case, can require checking a factorial number of paths, which could take an extremely long time. In order to facilitate this process, we employ a graph pruning approach. In the simulations of Section 6, we have used the pruned graph as an input to the exact algorithm. This has drastically reduced the amount of time required in the worst case, while slightly increasing the average time of the exact algorithm. The approach is based on the observation that two regenerator nodes that are directly connected by a link whose
physical impairment is less than the threshold $\Delta$ can be merged to form a ‘super regeneration node’. The supernode replaces the two nodes in the graph as follows:

- The supernode inherits all the links of either nodes except the link between them.
- In order to maintain a simple graph, if both nodes have the same neighbor, only the link with the smaller physical impairment is inherited by the supernode.

This process can be recursively continued until all the nodes that are reachable from each other using only regenerator nodes form a supernode. It can be done by randomly choosing a regenerator node as a root node and finding the shortest ‘regenerator-nodes-path-tree’, which contains regenerator nodes that are reachable from the root node using only regenerator nodes. We define a ‘cluster’ as the maximal set of regenerator nodes that are reachable from each other using only regenerator nodes. Thus, a regenerator node can belong to exactly one cluster. The example in Figure 9(a) has two clusters: one containing nodes 1, 3 and 4, and another containing nodes 6 and 7. Figure 9(b) shows the pruned graph.

![Graph](image)

Figure 9: An example for graph pruning where the request is $(s, d, 10)$. The shaded nodes in the graph are the regenerator nodes. (a) The original graph with two clusters. (b) The pruned graph with two supernodes.

Note that the pruning is intended to check the presence of a feasible path. However, the path obtained in the pruned graph is not necessarily the optimal path. In addition, if a feasible path does not exist in the pruned graph, there is no feasible path in the original graph and vice versa.
5.3 Exact Impairment-Aware Routing Algorithm (EIARA)

The Exact Impairment-Aware Routing Algorithm (EIARA) is described as follows. At each node, EIARA stores sub-paths, their corresponding lengths, and the set of regeneration nodes along the path. When comparing sub-paths, we use its predicted end-to-end length, instead of the length of the sub-path. This approach is intended to avoid unnecessarily scanning of the search space that is farther from the destination node.

To prevent, if possible, the more expensive operations in the latter parts of the code, EIARA calls the function loop_path in Line 1. If loop_path fails to find a path, then EIARA exits. However, if loop_path returns a path, this path is returned as a feasible solution by EIARA only if it does not contain loops. Hence, the function check_loop in Line 5 checks whether the path contains any loop. In Lines 11-13, dijkstra_all computes the shortest paths tree for each node \( v \in \mathcal{N}_R \cup \{ d \} \). For each node \( v \in \mathcal{N}_R \cup \{ d \} \), the distances of all nodes in the shortest paths tree rooted at \( v \) are stored in the array \( rdist[v] \). These distance values are then used in Lines 14-18 to obtain nearest_reg[u] and lowerbound[u] for each node \( u \in \mathcal{N} \), which are the distances of node \( u \) to its nearest regeneration node (or destination node) and the destination node, respectively. In addition counter[u], which represents the number of paths stored at each node \( u \), is set to zero. In Line 19, the queue \( Q \) is initialized to an empty set. The path counter of node \( s \) (i.e., counter[s]) is incremented in Line 20, and in Line 21 the path that contains only node \( s \) is inserted into the queue with a length of 0.

Lines 23–43 search for the solution as long as the queue \( Q \) is not empty (otherwise there is no feasible path). In Line 24, extract-min extracts the minimum length path (in terms of the predicted distance from the source node to the destination node along a path), and returns \( u[i] \), which is the \( i \)-th path stored in the queue at node \( u \). If node \( u \), which corresponds to the extracted path \( u[i] \), is the destination node, then path \( u[i] \) is returned as a solution. The path is reconstructed by backtracking through the predecessor list \( \pi \). If node \( u \) is not the destination node, each node adjacent to node \( u \) is considered in Lines 27-42. In Line 29, the function backtrack returns true if node \( v \) has already been encountered along this path, and false otherwise. In Line 30, look_ahead_a, which is the predicted length from the last regenerator node along the current path (i.e., \{u[i], v\}) to the nearest regenerator node of node \( v \) is computed. In Line 31, look_ahead_d, which is the predicted end-to-end length (i.e., source to destination node) of the current path is computed. If a cycle is not detected along the current path and the values of look_ahead_a and look_ahead_d do not exceed \( \Delta \) and \((R+1)\Delta\), respectively, the path is inserted into the queue in Line 33. The corresponding information associated with the new path, i.e. \( r \) (the length of the path), \( r_1 \) (the distance since the last regenerator node), \( \pi \) (the predecessor list) are assigned in Lines 34-36. If node \( v \) is a regenerator node, the length since the last regenerator node along the current path is set to zero in Lines 37-38. This does not mean that this regenerator node will necessarily be used along the current path. Instead, after EIARA finds the final solution,
Algorithm 4 EIARA($G, s, d, N_R, \Delta$)

1: $P_{s \rightarrow d} \leftarrow$ LOOP_PATH($G, s, d, N_R, \Delta$)
2: if $P_{s \rightarrow d} = NULL$ then
3: STOP $\rightarrow$ return no path found!
4: else
5: $loop \leftarrow$ CHECK_LOOP($P_{s \rightarrow d}$)
6: if (not loop) then
7: STOP $\rightarrow$ return $(P_{s \rightarrow d})$
8: end if
9: end if
10: 
11: for each $v \in N_R \cup \{d\}$ do
12: DIJKSTRA_ALL($G, v, rdist[v]$)
13: end for
14: for each $u \in N$ do
15: counter[$u$] $\leftarrow 0$
16: nearest_reg[$u$] $\leftarrow \min_{v \in N_R \cup \{d\}} \{rdist[v][u]\}$
17: lowerbound[$u$] $\leftarrow rdist[d][u]$
18: end for
19: queue $Q \leftarrow \emptyset$
20: counter[$s$] $\leftarrow$ counter[$s$] + 1
21: INSERT($Q, s$, counter[$s$], 0)
22: 
23: while ($Q \neq \emptyset$) do
24: EXTRACT-MIN($Q$) $\rightarrow u[i]$
25: if ($u = d$) then
26: STOP $\rightarrow$ return $u[i]$
27: else
28: for each $v \in Adj[u]$ do
29: cycle $\leftarrow$ BACKTRACK($v, u[i]$)
30: $look\_ahead_{a} \leftarrow r_{1}(u[i]) + r(u, v) + nearest\_reg[v]$
31: $look\_ahead_{b} \leftarrow r(u[i]) + r(u, v) + lowerbound[v]$
32: if (not(cycle) and $look\_ahead_{a} \leq \Delta$ and $look\_ahead_{b} \leq (R + 1) \ast \Delta$) then
33: INSERT($Q, v$, counter[$v$], $look\_ahead_{b}$)
34: $r[v[\text{counter}[v]]] \leftarrow r(u[i]) + r(u, v)$
35: $r_1[v[\text{counter}[v]]] \leftarrow r_1(u[i]) + r(u, v)$
36: $\pi[v[\text{counter}[v]]] \leftarrow u[i]$
37: if ($v \in N_R$) then
38: $r_1[v[\text{counter}[v]]] \leftarrow 0$
39: end if
40: end if
41: end for
42: end if
43: end while
it identifies regenerator nodes where regeneration is absolutely essential (i.e.,
the path is no more feasible without regeneration at these nodes). Then, the
solution path is returned along with its regenerator nodes list. Since EIARA is
essentially a brute-force approach that only prunes paths from the search-space
(via the look-ahead concept) that are provably infeasible, EIARA is guaranteed
to be exact.

The complexity of EIARA can be computed as follows. We will not explic-
licitly list $O(1)$ operations. In Line 1, algorithm loop_path has a complexity of
$O(RL + RN \log N)$, and the operations in Lines 11 – 13 have the same com-
plexity. Operations in Lines 14-18 have a total complexity of $O(RN)$. Let
$k_{\text{max}}$ be the maximum number of paths that are stored at any node. Hence,
the queue $Q$ contains at most $k_{\text{max}}N$ paths. When using a Fibonacci or Rel-
xaxed heap to structure the queue, selecting the minimum length path takes at
most $O(\log(k_{\text{max}}N))$ time [6]. Since each node can be selected at most $k_{\text{max}}$
times from the queue, the extract-min function in Line 24 takes at most
$O(k_{\text{max}}N \log(k_{\text{max}}N))$ time. Constructing the path in Line 26 takes at most
$O(N)$ time. The for loop starting in Line 28 is invoked at most $k_{\text{max}}$ times
for each side of each link in the graph, resulting in $O(k_{\text{max}}L)$ time. The back-
track function in Line 29 takes $O(N)$ time. Thus, the total running time of
Lines 23-43 is $O(k_{\text{max}}N \log(k_{\text{max}}N) + k_{\text{max}}LN)$. Combining the running times
of all the operations in the EIARA algorithm, results in

$$C_{\text{EIARA}} = O(RN \log N + k_{\text{max}}N \log(k_{\text{max}}N) + k_{\text{max}}LN)$$

(1)

5.4 Heuristics

In this section, we provide two heuristics, whose performance is evaluated in
Section 6. Our first heuristic is named TIARA, i.e., Tunable Impairment-Aware
Routing Algorithm, and it is identical to EIARA except that the maximum
number of subpaths $k_{\text{max}}$ that can be stored at a node is now bounded by a
fixed $k$ that is part of the input\(^3\). If $k = 1$, as set in the simulations, the
complexity of TIARA is $O(RN \log N + LN)$. The second heuristic is called the
Loop Avoidance Heuristic LAH.

Algorithm LAH($G, s, d, N_R, \Delta$), as in algorithm PIARA($G, s, d, N_R, \Delta$) given
earlier, computes the shortest paths between the regenerator nodes (including
$s$ and $d$) before creating graph $G'$ in step 2. The difference is that LAH tries to
avoid loops by assigning link weights in $G'$ that reflect the “criticality” of links,
which in this case is set equal to the number of other paths it overlaps with.
Other measures of criticality could also be used.

\(^3\)Our multi-constrained path selection heuristic TAMCRA is analogously derived from its
exact counterpart SAMCRA [18].
Algorithm 5 \( LAH(G, s, d, \mathcal{N}_R, \Delta) \)

1. For each pair of nodes \( i, j \in \mathcal{N}_R \cup \{s, d\} \) and \( i \neq j \), find the shortest path \( P^*_{i \rightarrow j} \).

2. Create a graph \( G' \) whose nodes belong to \( \mathcal{N}_R \cup \{s, d\} \) and there is a link \((i, j)\) in \( G' \) if the shortest path \( P^*_{i \rightarrow j} \) was feasible in the original graph \( G \), i.e., \( r(P^*_{i \rightarrow j}) \leq \Delta \).

3. Set the cost of link \((i, j)\) in \( G' \) equal to the number of other paths that \( P^*_{i \rightarrow j} \) shares a segment (i.e., a link or more) with.

4. Find the shortest path from \( s \) to \( d \) in \( G' \) and substitute each link \((i, j)\) in the shortest path with the corresponding subpath \( P^*_{i \rightarrow j} \) in \( G \) to obtain the solution.

5. Return the path if it is loop-free, else return fail.

6 Simulation Results

The following simulation results compare the performance of the two heuristic algorithms and the exact algorithm for random networks and lattice networks. We present three types of simulations:

1. Head-to-head comparisons of the three algorithms under dynamic traffic (requests). As each request arrives, all the three algorithms are run and their results are compared. If the request is accepted, the path returned by the exact algorithm is assigned to the request before proceeding to the next request. Thus, the network changes dynamically since the links associated with already existing requests are dropped from the network.

2. Independent comparison of the three algorithms under dynamic traffic. In this scenario, the three algorithms are run independently and requests are allocated separately by each algorithm.

3. Case-by-case comparisons of the three algorithms. We create thousands of graphs and for each graph run the algorithms for only a single request. The results are averaged over all graphs.

We have performed extensive simulations on random and lattice networks, where the link weights are uniformly distributed in the range \((0, 1]\). The source and destination nodes are randomly assigned. The regenerator nodes are also placed randomly.

In order to make a fair comparison, the results given for the path length, regenerator count are for requests that are accepted by all the three algorithms. For the computation times, we present both the average for requests that are accepted by all the three algorithms and the overall average of all requests. The
results for the lattice networks have been worse than for the random graphs, when the network size and link weight distribution are the same. The reason is that a larger expected hopcount in lattice networks increases the probability that the impairment threshold will be violated. Our results have indicated that EIARA is generally fast when a feasible path exists, but in some cases (see Figures 14(d), 19(d) and 24(c)) it can take a long time to decide that no solution exists. This is because in the worst case it has to check a factorial number of paths.

6.1 Random Networks

6.1.1 Different Arrival Rates

Figures 10(a)-(e) show results of the head-to-head comparisons of the three algorithms under dynamic traffic with a Poisson arrival rate and exponential holding time of 1 for a random network of $N = 50$ and $p = 0.1$ by varying the arrival rate of requests.

A higher arrival rate results in a higher traffic intensity and a quicker occupation of the regenerator capacities, which in turn decreases the acceptance ratio.

Figures 11(a) and (b) show the independent performance of the three algorithms under dynamic traffic. It can be seen that at times, the heuristic algorithms (especially TIARA) have a better acceptance ratio than the EIARA. This is because, EIARA, which returns an optimal solution for a single request, may not necessarily be optimal for the overall dynamic traffic.

6.1.2 Different Physical Impairment Thresholds

Figures 12 (a) and (b) show results of the case-by-case comparisons of the three algorithms for random networks of $N = 50$ and $p = 0.1$ by varying the physical impairment threshold $\Delta$. In these simulations, 10000 random networks were used for each threshold value. For each random network, there is a single request and the three algorithms are run.

An increase in the impairment threshold leads to a relaxation of the problem and consequently increases the probability that a feasible path exists (and is found).

6.1.3 Different Number of Nodes

Figure 13 shows the case-by-case acceptance ratio comparison of random networks of $p = 0.1$ by varying their number of nodes ($N$).
Figure 10: Head-to-head comparison of (a) the acceptance ratio, (b) average path length, (c) average number of used regenerators, and the average times (in ms) of (d) accepted requests and (e) all requests of EIARA, TIARA with $k = 1$, and LAH for different arrival rates of flows. (A random network of $N = 50$ and $p = 0.1$, number of flows = 10000, $\Delta = 1$, number of regenerator nodes = 15, regenerators per node = 2)
Figure 11: Independent comparison of (a) the acceptance ratio and (a) average times of accepted requests (in ms) of EIARA, TIARA with $k = 1$, and LAH for different arrival rates of flows. (A random network of $N = 50$ and $p = 0.1$, number of flows = 10000, $\Delta = 1$, number of regenerator nodes = 15, regenerators per node = 2)

Figure 12: Case-by-case comparison of the (a) acceptance ratio and (b) average path length of EIARA, TIARA with $k = 1$, and LAH for different random networks (of $N = 50$ and $p = 0.1$) and different impairment thresholds ($\Delta$). (Number of networks = 10000, number of regenerator nodes = 15)
Figure 13: Case-by-case comparison of the acceptance ratio of EIARA, TIARA with \( k = 1 \), and LAH for different random networks and different number of nodes \((N)\). (Number of networks = 10000, number of regenerator nodes = 15, \( \Delta = 1 \), \( p = 0.1 \))

Under a fixed number of regenerator nodes, increasing the network size will result in longer expected path lengths and a higher probability of violating the threshold.

6.1.4 Different Number of Regenerator Nodes

Figures 14(a)-(d) show results of the case-by-case comparisons of the three algorithms for random networks of \( N = 50 \) and \( p = 0.1 \) by varying the number of regenerator nodes \( R \). From Figure 14(d), it can be seen that EIARA can sometimes take a very long time compared to the heuristic algorithms because it has to check more paths before deciding a feasible path does not exist.

In agreement with our results for different network sizes, we again observe that an increase in the ratio of the number of regenerators versus the number of nodes \( \frac{R}{N} \), an increase in \( \frac{R}{N} \) results in an increase of the acceptance ratio.

6.2 Lattice Networks

For the simulations in the lattice networks, similar assumptions were made as for the random networks. We can see that the acceptance rate in the Lattice networks is lower than for the random networks, which was expected since paths in Lattice networks are generally longer in hopcount than in random networks. A larger hopcount increases the chance of violating the physical impairment.
Figure 14: Case-by-case comparison of (a) the acceptance ratio, (b) average path length, and average times (in ms) of (c) accepted and (d) all requests of EIARA, TIARA with $k = 1$, and LAH for different random networks (of $N = 50$ and $p = 0.1$) and different number of regenerator nodes ($R$). (Number of networks = 10000)
threshold (assuming an equal amount of nodes and regenerators in these different classes of networks).

6.2.1 With Node Pruning

In this section, we give results for lattice networks, where there is a possibility of pruning since two nodes can be adjacent to each other.

Different Arrival Rates  Figures 15(a)-(c) show results of the head-to-head comparisons of the three algorithms under dynamic traffic with a Poisson arrival rate and exponential holding time of 1 for a lattice network of $N = 64$ by varying the arrival rate of requests.

Figures 16(a) and (b) show the independent performance of the three algorithms under dynamic traffic for a lattice network.

Different Physical Impairment Thresholds  Figure 17 shows results of the case-by-case acceptance ratio comparison of the three algorithms for lattice networks of $N = 64$ by varying the physical impairment threshold $\Delta$.

![Graph showing acceptance ratio versus impairment threshold](image)

Figure 17: Case-by-case comparison of the acceptance ratio of EIARA, TIARA with $k = 1$, and LAH for different lattice networks (of $N = 64$) and different impairment thresholds ($\Delta$). (Number of networks = 10000, number of regenerator nodes = 20)

Different Number of Nodes  Figure 18 shows the case-by-case acceptance ratio comparison of lattice networks by varying their number of nodes ($N$).
Figure 15: Head-to-head comparison of (a) the acceptance ratio, (b) the average path length, and (c) the average times (in ms) of EIARA, TIARA with k=1, and LAH for different arrival rates of flows. (A lattice network of $N = 64$, number of flows = 10000, $\Delta = 1$, number of regenerator nodes = 20, regenerators per node = 2)
Figure 16: Independent comparison of (a) the acceptance ratio and (b) average times (in ms) of EIARA, TIARA with $k = 1$, and LAH for different arrival rates of flows. (A lattice network of $N = 64$, number of flows = 10000, $\Delta = 1$, number of regenerator nodes = 20, regenerators per node = 2)

Figure 18: Case–by-case comparison of the acceptance ratio of EIARA, TIARA with $k = 1$, and LAH for different lattice networks and different number of nodes ($N$). (Number of networks = 10000, number of regenerator nodes = 20, $\Delta = 1$).
Different Number of Regenerator Nodes  Figures 19(a)-(d) show results of the case-by-case comparisons of the three algorithms for lattice networks of $N = 64$ by varying the number of regenerator nodes $R$.

6.2.2 Without Node Pruning

In this section, the regenerator nodes are placed in such a way that no two regenerator nodes are neighbors. Therefore, node pruning is not possible.

Different arrival rates  Figures 20(a)-(c) show results of the head-to-head comparisons of the three algorithms under dynamic traffic with a Poisson arrival rate and exponential holding time of $1$ for a lattice network of $N = 49$ by varying the arrival rate of requests.

Figures 21(a) and (b) show the independent performance of the three algorithms under dynamic traffic for a lattice network without node pruning.

Different impairment thresholds  Figure 22 shows results of the case-by-case acceptance ratio comparison of the three algorithms for lattice networks of $N = 49$ by varying the impairment threshold $\Delta$.

Different number of nodes  Figure 23 shows the case-by-case acceptance ratio comparison in lattice networks of different sizes ($N$).

Different number of regenerator nodes  Figures 24(a)-(c) show results of the case-by-case comparisons of the three algorithms for lattice networks of $N = 49$ by varying the number of regenerator nodes $R$.

In general, the simulations show that although $LAH$ is somewhat faster than $TIARA$, $TIARA$ is also fast and always outperforms (often considerably) $LAH$. Moreover, the quality of $TIARA$’s solutions are quite close to the exact solutions of $EIARA$. Hence, $TIARA$ is our preferred choice, offering close-to-optimal performance within reasonable computational complexity.

7 Conclusions

In optical networks, physical impairments, such as noise and signal distortions, degrade the quality of the signal. These impairments become more severe with distance and bit rate, unless the signal is regenerated timely. Regenerators are costly and therefore generally sparsely deployed. In this paper we have studied the problem of selecting paths that meet a certain impairment constraint. We have provided a polynomial-time algorithm for the case that paths may contain loops. If loops are not permitted, we have shown that the problem is strongly NP-complete and subsequently provided exact and heuristic solutions. Through simulations we have demonstrated that our heuristic $TIARA$ is computationally efficient and offers close-to-optimal solutions.
Figure 19: Case-by-case comparison of (a) the acceptance ratio, (b) the average path length, and the average times (in ms) (c) accepted requests and (d) all requests of EIARA, TIARA with $k = 1$, and LAH for different lattice networks (of $N = 64$) and different number of regenerator nodes ($R$). (Number of networks = 10000)
Figure 20: Head-to-head comparison of (a) the acceptance ratio, and the average times (c) accepted requests and all requests of the three algorithms for different arrival rates of flows. (Lattice network, $N = 49$, number of flows = 10000, $\Delta = 1$, $R = 24$, regenerators per node = 2).
Figure 21: Independent comparison of (a) the acceptance ratios and (b) the average times (in ms) different arrival rates of flows. (Lattice network, \(N = 49\), number of flows = 10000, \(\Delta = 1\), \(R = 24\), regenerators per node = 2).

Figure 22: Case-by-case comparison of the acceptance ratio for lattice networks \((N = 49)\) and different impairment thresholds \((\Delta)\). (Number of networks = 10000, \(R = 12\)).
Figure 23: Case-by-case comparison of the acceptance ratio for lattice networks and different number of nodes (N). (Number of networks = 10000, R = 16, Δ = 1).

An important subject for future work is to consider the design perspective, namely to investigate where to best place the regenerators.

References


Figure 24: Case-by-case comparison of (a) the acceptance ratio, and the average times of (a) accepted requests and (c) all requests for lattice networks ($N = 49$) and different number of regenerator nodes ($R$). (Number of networks = 10000 and $\Delta = 1$).


