Delft University of Technology

Faculty of Electrial Engineering, Mathematics and Computer Science
Network Architectures and Services

Impairment-Aware QoS Routing in Translucent Optical Networks

Ebisa Olana Negeri
(1386514)

A thesis submitted in partial fulfillment of

Master of Science

Committee Members:

Supervisor:	Dr. Ir. Fernando Kuipers
Mentor:	Anteneh Ayalew Beshir
Others:	Prof. Dr. Ir. Piet Van Mieghem, Dr. Ir. Christian Doerr, Dr. Ir. Anthony Lo

June 29, 2009
M.Sc. Thesis No: PVM 2009-055

Abstract

Wavelength-division-multiplexed (WDM) optical networks are commonly used to transport huge amount of traffic in long-haul and metro/regional networks. In these networks, the optical signals deteriorate due to the physical impairments they encounter as they traverse multiple links. This necessitates regeneration of the signals at the intermediate nodes so that the signals will reach the destination with an acceptable level of quality. In translucent optical networks, the regenerators are sparsely placed in the network. Some applications that are transported over these networks require a guaranteed end-to-end quality of service (QoS). The QoS routing in these networks involves two tasks: guaranteeing the end-to-end QoS requirement, and making sure that the signal quality will be acceptable at the destination by considering the physical impairments.

In this thesis, we present physical impairment-aware QoS routing algorithms in translucent optical networks. We have proposed exact and heuristic algorithms that aim at optimally satisfying the QoS requirements, and minimizing the number of regenerators used along the selected path. The attractive feature of our algorithms is that they incorporate both the physical impairments and the regenerator assignment into the path computation process. As a result, the paths are computed efficiently. The experimental results show that each of our algorithms has its own aspect of fitness where it should be the best choice over the others.

Acknowledgements

In the first place, my unparalleled praise goes to God for His providence.

I would like to record my gratitude to my advisor, Dr. Fernando A. Kuipers for his continuous support and his invaluable advices that have enormous contribution to my work. His deep insight into problems has exceptionally inspired and enriched my growth as a student.

My earnest appreciation goes to my daily supervisor, Anteneh Ayalew for his unreserved all-time assistance in various ways. In addition to providing me with the thesis title, he has guided me, provided me with precious comments on my work, and encouraged me in difficulties. I am highly indebted to him.

I would like to extend my courteous thanks to Professor Piet Van Mieghem. It gives me great pleasure to do this thesis under his academic supervision.

Where would I be without my family? Words fail me to express my appreciation for the love and encouragement they have shown me to this day.

Table of Contents

Abstract **iii**

Acknowledgements **v**

Acronyms **ix**

1 Introduction **1**

1-1 WDM Optical networks 1

1-1-1 Routing and Wavelength Assignment 2

1-1-2 Physical Layer Impairments 4

1-1-3 QoS Routing 5

1-2 Related Work 6

1-3 Objectives 8

1-4 Organization of the Thesis 9

2 Impairment-Aware QoS Routing (IQR) **11**

2-1 Network Model 11

2-2 Notations 13

2-3 Problem Definition 14

2-4 General Approach 15

2-5 Exact Algorithm 15

2-5-1 QoS Optimization 17

2-5-2 Regenerator-Count Minimization 26

2-6 Heuristic Algorithms 32

2-6-1 Tabu-Search Overview 32

2-6-2 The Tabu-search Based Heuristic Algorithm 33
2-6-3 QoS Optimization 35
2-6-4 Regenerator-Count Minimization 41

3 Simulation 45
3-1 Simulation Environment 45
3-2 Simulation under Totally Loose Physical Constraints 46
3-3 Simulation under Totally Loose QoS Constraints 49
3-4 Simulation under Tight QoS Constraints 55
3-5 Simulation under Medium QoS Constraints 59
3-6 Simulation under Dynamic Traffic 63

4 Discussion and Conclusion 65
4-1 Discusions 65
4-2 Conclusions 67
4-3 Future Work 68

A Network Topologies 75
A-1 The ARPANET Network 75
A-2 A Lattice Network 76

B Additional Results 77
B-1 Totally loose physical constraints 77
B-2 Totally loose Qos constraints 81
B-3 Tight Qos constraints 88
B-4 Medium Qos constraints 91
B-5 Tunable algorithms with different values of k 95
B-6 Dynamic traffic 97

Acronyms

ASE	Amplifier Spontaneous Emission
BER	Bit Error Rate
BF	Best Fit
EDFA	Erbium Doped Fiber Amplifier
EIOQRA	Exact Impairment-aware Optimal QoS Routing Algorithm
EIQRRM	Exact Impairment-aware QoS Routing with Regenerator Minimization
FF	First Fit
IOQR	Impairment-aware Optimal QoS Routing
IOQRA	Impairment-aware Optimal QoS Routing Algorithm
IQR	Impairment-aware QoS Routing
IQRRM	Impairment-aware QoS Routing with Regenerator Minimization
OEO	Optical-Electronic-Optical
OXC	Optical Cross-Connects
PMD	Polarization Mode Dispersion
RF	Random Fit
RWA	Routing and Wavelength Assignment

SAMCRA	Self-Adaptive Multi-Constrained Routing Algorithm
TIOQRA	Tunable Impairment-aware Optimal QoS Routing Algorithm
TIQRRM	Tunable Impairment-aware QoS Routing with Regenerator Minimization
TS	Tabu Search
WDM	Wavelength-Division-Multiplexed

Chapter 1

Introduction

1-1 WDM Optical networks

WDM optical networks are widely used in long-haul and metro/regional networks. Their architecture is evolving from traditional opaque networks towards all-optical (i.e. transparent) networks. In transparent all-optical networks, the signal is transmitted in the optical domain from the source to the destination node without undergoing any optical-electronic-optical (OEO) conversions. In opaque networks, the optical signal carrying traffic undergoes an OEO conversion at every switching or routing node. Practically, the transmission reach of optical signals is limited (e.g. 2000-2500 km) [2]. In order to go beyond this transparent reach of optics, signal regeneration is required at intermediate nodes to re-amplify, re-shape and re-time the optical signal, which are collectively known as the 3R regeneration. Even though in principle optical 3R regeneration can be accomplished completely in the optical domain (e.g. [5, 6]), only electrical 3R regenerators are currently economically viable [3]. Hence, signal regeneration involves OEO conversions which disrupt the transparency of the signal.

The OEO process increases the cost of the signal transmission due to several factors such as the number of regenerators required in the network, the dependency of the conversion process on the connection line rate and also on the modulation format [1]. With each OEO node are associated scalability issues related to cost, space requirements, power consumption and heat dissipation. Hence, for large size of opaque networks, network designers and architects have to consider more electronic terminating and switching equipments, which presents challenges in cost, heat dissipation, power consumption, required physical space, and operation and maintenance costs. Due to the lack of practical all-optical regenerators, these issues are addressed by the intermediate optical network architectures, which are known as translucent networks [7]. Translucent network architectures have been proposed as a compromise between opaque and all-optical networks. This approach places regenerators sparsely and strategically to maintain the acceptable level of signal quality from the source to its destination. Hence, much of the required electronic processing is eliminated and a signal is allowed to remain in the optical domain for much of its path. Keeping the signals in the optical domain brings a significant cost reduction due to removal of electronic processing equipments [8]. This removal of electronic devices further reduces power consumption, heat dissipation and space requirements. Our study focuses on translucent networks.

1-1-1 Routing and Wavelength Assignment

The rapid advancement and evolution of optical technologies makes it possible to move beyond point to point WDM transmission to an all-optical backbone network that can take full advantage of the available bandwidth. Such a network consists of a number of optical cross-connects (OXCs) arranged in some arbitrary topology, and its main function is to provide interconnection to a number of edge devices in WDM backbone networks. Each OXC can switch the optical signal coming in on a wavelength of an input fiber link to the same wavelength in an output fiber link. The OXC may also be equipped with convertors that permit it to switch the optical signal on an incoming wavelength of an input fiber to some other wavelength on an output link. The optical signal is transported over a lightpath, which is an optical communication channel established over the network of OXCs which may span a number of fiber links. If no wavelength convertors are used, a lightpath is associated with the same wavelength on each hop. Using convertors, a different wavelength on each hop may be used to setup a lightpath. Thus a lightpath is an end-to-end optical connection established between two edge devices attached to the optical backbone [10, 11].

Establishing an optical connection in WDM optical networks involves both routing (selecting a suitable path) and wavelength assignment (allocating an available wavelength for the connection). For a given set of connection requests, the problem of setting up lightpaths by routing and assigning a wavelength to each connection with the goal of maximizing the number of optical connections is referred to as Routing and Wavelength-Assignment (RWA) problem [9]. The RWA problem is significantly more difficult than the routing problem in electronic networks. The additional complexity arises from the fact that routing and wavelength assignment are subject to the following two constraints:

1. *Wavelength continuity constraint:* if no regenerators are used, a lightpath must use the same wavelength on all the links along its path from the source to the destination node.

2. *Distinct wavelength constraint:* all lightpaths using the same link must be allocated distinct wavelengths.

The wavelength continuity constraint may be relaxed if the OXCs are equipped with wavelength convertors [12]. A wavelength converter is a single input/output device that converts the wavelength of an optical signal arriving at its input port to a different wavelength as the signal departs from its output port, but otherwise leaves the optical signal unchanged. In general, wavelength continuity constraint is imposed on each lightpath since optical wavelength convertors remain too costly.

The effect of wavelength continuity constraint can be represented by replicating the network into as many copies as the number of wavelengths. If wavelength i is selected for a lightpath, the source and destination node communicate over the i^{th} copy of the network. Thus, finding a path for a connection may potentially involve W routing problems for a network with W wavelengths, one for each copy of the network.

Given a set of candidate wavelengths satisfying the wavelength continuity constraint, the wavelength selection can be performed in various ways, such as the *first fit(FF), best fit(BF),* and *random fit(RF)*. In the first fit approach, the first non-occupied wavelength that satisfies the connection requirements is selected [16, 17]. The best fit approach tries to look through all of the candidate wavelengths so as to find the most appropriate one [17, 18]. In the random fit approach, a wavelength is randomly chosen among the available wavelengths [19].

The RWA problem can be cast in numerous forms. The different variants of the problem, however, can be classified into two broad versions: a static RWA, whereby the traffic requirements are known in advance, and a dynamic RWA, in which a sequence of lightpath requests arrive in some random fashion.

Static RWA

If the traffic patterns in the network are reasonably well-known in advance and any traffic variations take place over long time scales, the most effective technique for establishing lightpaths between client subnetworks is by formulating and solving a static RWA problem. Therefore, static RWA is appropriate for provisioning a set of semi-permanent connections. Since these connections are assumed to remain in place for relatively long periods of time, it is worthwhile to attempt to optimize the way in which the network resources are assigned to each connection.

Dynamic RWA

During real-time network operation, edge nodes submit to the network their requests for lightpaths to be set up as needed. Thus connection requests are initiated in some random fashion. Depending on the state of the network at the time of a request, the available resources may or may not be sufficient to establish a lightpath between the corresponding source-destination node pair. The state evolves randomly in time as new lightpaths are admitted and existing lightpaths are released. Thus, each time a request is made, an algorithm must be executed in real time to determine whether it is feasible to accommodate the request, and, if so, to perform routing and wavelength assignment. If a request for a lightpath cannot be accepted because of lack of resources, it is blocked.

Previous studies that investigated the RWA problem are summarized in [9], and the problem is known to be NP-Complete [9].

1-1-2 Physical Layer Impairments

Optical signals encounter many impairments that affect the signal intensity level, as well as its temporal, spectral and polarization properties as they traverse the optical fiber links and also propagate through the optical components [20]. These impairments are collectively known as *physical impairments*. The physical impairments accumulate noise and signal distortions along the physical path which degrade the quality of the received signal. The signal degradation may lead to an unacceptable bit error rate (BER) particularly for high bit rates and when the signal travels long distances. Therefore, physical impairment-aware routing is required in optical networks.

Physical layer impairments can be categorized into *linear* and *non-linear impairments*. Linear impairments are independent of the signal power and affect each of the wavelengths (optical channels) separately. Consequently, they can be treated as constraints associated to links. On the other hand, non-linear impairments affect not only each optical channel individually but they also cause disturbance and interference between them [21, 22]. Because of the disturbance and interference they introduce between the optical channels, non-linear impairments are notably more complex than linear impairments. Especially, non-linear impairments intensely depend on the current allocation of wavelengths on a given fiber. Thus, they depend not only on the the physical topology, but also on the current state of the allocated lightpaths [24]. Consequently, the establishment of a new lightpath is influenced by the already established ones. Likewise, a newly established lightpath may affect the transmission properties of the previously established ones.

In this thesis, we consider only linear impairments. It is necessary to consider multiple impairments in the routing process because the individual impairments depend on different properties of the optical network. For example, PMD (polarization mode dispersion) and ASE (Amplifier spontaneous emission) noises, that have been identified in an Internet Engineering Task Force (IETF) draft as the two major linear impairments that can practically be used in constraint-based optical routing [23][27], depend on different properties of the network. PMD results from the pulse spread in the frequency domain due to different polarizations of the optical signal traveling at different velocities, which normally travel at the same velocity. For a *regeneration segment*, which is the segment between two consecutive nodes where regenerations take place along a lightpath, the PMD is given as [27]:

$$PMD_{Reg\_Seg} = \sqrt{\sum_{i \in Reg\_Seg} {D_{PMD}}^2(i) l(i)} \tag{1-1}$$

where Reg_Seg is the regeneration segment, i is the link index, $D_{PMD}(i)$ is the fiber dispersion parameter at the i^{th} optical link, and $l(i)$ is the length/distance in km of the i^{th} link. The typical values of $D_{PMD}(i)$ range from 0.1 to 0.5 $ps/\sqrt{Km}$ [27]. The maximum value for PMD_{Reg_Seg} is expressed as [27]:

$$PMD_{Reg\_Seg} \leq \frac{\alpha}{B} \tag{1-2}$$

where α is the maximum dispersion fraction, and B is the digital bit error rate of the signal. A typical value for α is 0.1, thus $B \leq 1$. In order to have a linear relationship, we use $PMD^2_{Reg\_Seg}$ as a parameter. Thus,

$$\sum_{i \in Reg\_Seg} D^2_{PMD}(i) l(i) \leq \left(\frac{\alpha}{B}\right)^2 \tag{1-3}$$

On the other hand, more ASE noise will be accumulated as more EDFAs (erbium doped fiber amplifiers) are traversed, because each EDFA stage adds its own component of ASE noise. The ASE noise power of an EDFA can be expressed as [27]:

$$P^{ASE}_{i,j} = n_{sp}(i,j)(G(\lambda_i, j) - 1) h v_i B_o \tag{1-4}$$

where $n_{sp}(i,j)$ is the spontaneous emission factor of the j^{th} EDFA on the i^{th} fiber link along a regeneration segment, $G(\lambda(i,j))$ is the saturated gain, $\lambda_k = c/v_i$ is the assigned wavelength, h is Planck's constant, c is the velocity of light, and B_o is the optical bandwidth. The ASE noise powers of different wavelengths along the same link are slightly different. If these differences are ignored, the ASE noise power on a regenerator segment is linearly related to the ASE noise power on individual links along that segment. The threshold value of the ASE power is expressed as [27]:

$$\sum_{i \in Reg\_Seg} \left(\sum_{j \in Link(i)} n_{sp}(i,j) G(\lambda(i,j) - 1) h v B_o \right) \leq P^{ASE}_{max} \tag{1-5}$$

where P^{ASE}_{max} is the threshold value of acceptable ASE noise, and λ and v are constants.

1-1-3 QoS Routing

Quality of Service (QoS) is the performance level of a service offered by the network to the user. The goal of QoS provisioning is to achieve a more deterministic network behavior, so that information can be better delivered and network resources can be better utilized. In WDM optical networks spanning large areas, an optical signal may traverse a number of intermediate nodes and long fiber segments. Each component along the path such as fibers, OXCs, and EDFAs are associated with some delay, cost, and reliability factors.

Different applications/end users need different levels of services and differ in how much they are willing to pay for the service they get. So, the network provider should provide different kinds of qualitative guarantees such as maximum delay, maximum cost, and minimum reliability to the users, depending on their requirements. As these services are route dependent, the RWA algorithm should find a route which satisfies the QoS requirements of the connection and best utilizes the network resources.

QoS can be classified into two categories: *qualitative requirements* and *functional requirements*. Some of the qualitative requirements of optical signal are transmission delay and and reliability of the components used along the path chosen for establishing a connection. Reliability of a component is the probability that the component functions correctly over a period of time [29]. An example of the functional requirements of optical signals is survivability. In WDM networks, the failure of a network component leads to the failure of all the paths traversing through that component. As each lightpath carries a huge volume of traffic, it is important that these networks are fault-tolerant. Fault-tolerance refers to the ability of the network to configure and reestablish communication upon failure. A fault-tolerant network is referred to as a survivable network. Several papers can be found in the literature on survivable WDM networks [29, 30, 31, 32]. The focus of this thesis is on additive and multiplicative qualitative QoS requirements.

The problem of QoS routing in electronic networks with multiple additive link weights is known to be NP-complete [33], and several heuristic, approximation, and exact algorithms have been suggested in the literature. A survey of these algorithms is provided in [34]. In translucent WDM optical networks, the presence of physical impairments complicates the problem even more. Thus, physical impairment aware QoS routing in WDM networks is also an NP-complete problem. In addition to satisfying the QoS requirements, a solution path to this problem should also meet the requirement that the physical impairment threshold is not exceeded on each regeneration segment along the path.

1-2 Related Work

Numerous RWA algorithms have been proposed in the literature for WDM optical networks. The great majority of these algorithms assume ideal physical layer conditions [9, 35, 36]. These algorithms are evaluated by using the blocking probability as a performance metric. A blocking event, called *wavelength blocking,* occurs when a lightpath cannot be setup due to shortage of a free route or a jointly free wavelength along the route.

Different approaches have been considered in the recent literature to provide the physical impairment awareness in RWA algorithms. In one approach, the route and the wavelength are computed in the traditional way without taking into account the physical impairments, and finally the selected lightpath is verified considering the physical layer impairments [37, 16, 39, 40, 18]. In this approach, other paths are recomputed if the candidate paths do not meet the physical impairments. The work presented in [37] comprises of a static regenerator placement and physical impairment aware RWA algorithm for dynamic traffic. After annotating each link with a Q-factor penalty, this algorithm computes paths between any combination of nodes under the constraint of a minimal Q value. In [16], the LERP (lightpath establishment and regenerator placement) algorithm is proposed as a physical impairment aware RWA and regenerator placement algorithm. In this algorithm, k alternative paths are first computed, and then the wavelength is assigned using the FF or RF method. Finally, the lightpath is tested with the Q values and regenerators are placed as required.

An algorithm that modifies the Bellman-Ford algorithm is proposed in [39]. In this algorithm, the minimum hop path is computed first with a certain cost limit. Then the path is checked at the destination if it satisfies the physical constraints. If the path fails the test, then another path is computed and checked until a path that satisfies the physical constraints is found. A similar approach is used in [18] where a shortest path is computed first among all available wavelengths and the path is verified with the physical impairment. The algorithm suggested in [40] first selects the wavelength of the path using the FF algorithm and then the shortest path is computed for the wavelength. Finally, the shortest path is verified with the physical constraint.

In the other approach, the physical layer impairment values are considered in the routing and/or wavelength assignment decisions [41, 42, 43, 45, 46, 47]. In some of the works involving this approach, the physical layer information is used as weight of the links in order to compute the minimum cost lightpath. The dynamic AQoS (Adaptive Quality of Service) routing algorithm proposed in [41] assigns routes based on real-time Q factor measurements. The algorithm computes shortest cost path or the k link disjoint shortest cost paths considering the Q value as the link cost. The algorithm also applies the CLC (Constrained Least Congested) approach in route selection to use the wavelengths efficiently. Depending on the network conditions, the final decision is taken by considering either the wavelength balancing efficiency or the Q factor value.

In the algorithm proposed in [42], the wavelength is initially selected by means of FF method without considering the physical impairments, and then a shortest path is computed for that wavelength considering the noise variance of the physical impairments as the link cost. Finally, if a lightpath causes a BER value to exceed a given threshold for the new lightpath or for other already established lightpaths, then it is discarded. In [43], an algorithm that takes the physical impairments into account in both routing and wavelength assignment processes is proposed. In this algorithm, the Q-Penalty values are used as the link costs to compute the k shortest routes; and the wavelength that maximizes the Q value is selected. Finally, the lightpath is verified if it satisfies another physical impairment.

In [45], k-shortest paths are computed by first removing the links that violate the ASE constraint. Then, a path that minimizes the number of hops is selected and afterwards the path is verified at the destination with other physical constraints. The algorithm proposed in [46] computes the shortest path without considering the physical impairments. In this algorithm, the physical impairments are considered in wavelength assignment and the final verification of the path. In [47], the suggested algorithm performs the physical constraint verification at each hop during the path computation process. Moreover, the selected path is verified with some physical impairments at the destination.

Most of the physical impairment aware RWA related works mentioned before focus on transparent optical networks. Even though there are few works on translucent optical networks, they do not integrate regenerator assignment in the path computation process of the routing algorithm. Most of them first compute a shortest path, and then check if the lightpath satisfies the considered constraint on the physical impairment; if the check fails, regeneration is introduced in the last regenerator node where the impairment constraint is respected. If the path fails after trying all possible regenerations, the second shortest path is computed and so on. Obviously, this is not an efficient approach. Further, these related works do not address QoS routing.

1-3 Objectives

In this thesis, the problem of physical impairment aware QoS routing is studied and solutions are provided. This thesis mainly differs from the previous works in that it incorporates both the physical impairments and the regenerator assignment in the path computation process. In addition, both the physical impairments and the QoS constraints are simultaneously addressed during the path computation process.

The 3R regenerators involve OEO conversions that disrupt the transparency of the signal. The OEO conversion process takes more time than switch- ing in the optical domain, thereby introducing delay into the signal. Thus, it is necessary to minimize the number of regenerators used along a path. On the other hand, the end users may require the optimal QoS. These two requirements lead to the two objectives of the thesis work.

The main objectives of the thesis are:

- To implement both exact and heuristic impairment aware QoS routing algorithms in translucent optical networks that try to optimally satisfy the QoS requirements, and investigate their performances.

- To implement both exact and heuristic impairment aware QoS routing algorithms in translucent optical networks that try to minimize the number of regenerators used along a path, and investigate their performances.

1-4 Organization of the Thesis

In chapter 2, the problem of impairment-aware QoS routing with QoS optimization and regenerator minimization objectives are addressed. We present both exact and heuristic algorithms that solve the problems.

In chapter 3, we present the simulation results of the algorithms developed in chapter 2. We also compare the relative performance of the algorithms.

Finally, in chapter 4 we give a general conclusion on the work done and make some recommendations for future work.

Chapter 2

Impairment-Aware QoS Routing (IQR)

In optical networks, the physical impairments deteriorate the quality of the signal as it traverses multiple links. Therefore, the QoS routing should be aware of the physical impairments. In this chapter, we study the problem of impairment aware QoS routing in translucent optical networks. The impairment-aware QoS routing problem is basically a multi-constrained routing problem with additional constraints on the physical impairments. Thus, the path selected as a solution to this problem should not only satisfy all the QoS constraints, but it should also meet the requirement that the physical impairments of each of the regeneration segments that make up the lightpath should not exceed their corresponding threshold values.

Unlike the QoS metrics that are accumulated from end to end along the path, the accumulated physical impairments are reset to zero when the signal is regenerated at intermediate regenerator nodes along the path. After regeneration, the accumulation of the physical impairments is resumed on the remaining links until the next regeneration takes place. This effect of regeneration on the physical impairments complicates the routing process compared to the routing in electronic networks. Since the multi-constrained path problem, which is proved to be NP-complete [33], is a reduced version of the impairment-aware QoS routing problem, the impairment-aware QoS routing problem is also NP-complete. In this chapter, we present exact and heuristic algorithms to solve the problem with two different objectives; namely, QoS optimization and regenerator-count minimization.

2-1 Network Model

In this thesis, we consider translucent optical networks, that are networks with sparsely placed OEO regenerators. A typical transmission system of a translucent optical transport

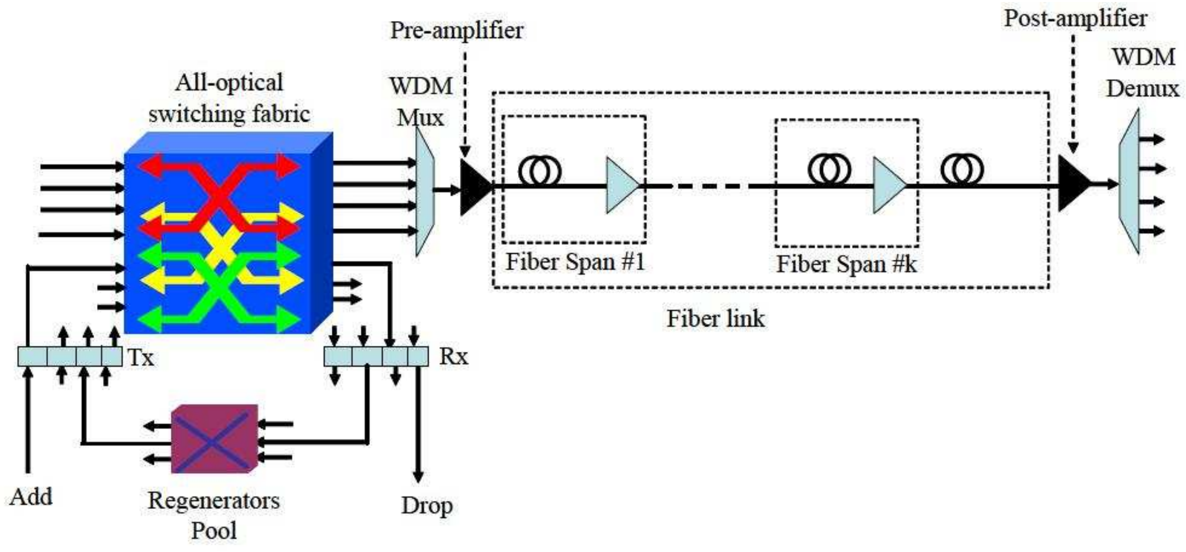


Figure 2-1: Some components that make up a translucent WDM optical network.

network (OTN) comprises of the components that are shown in Figure 2-1. In this model, a node includes an all-optical switch, transponders, optional 3R regenerators, pre- and post- amplifiers, multiplexers, and demultiplexers; whereas a link is a WDM line system incorporating fibers and amplifiers. Electrical signals are modulated onto distinct wavelengths by transponders at a given node. After these wavelengths are multiplexed, they are pre-amplified before being propagated through the WDM line. In order to subdue the fiber absorption losses, EDFAs are used at some points along the WDM line. Eventually, the signal is post-amplified and demultiplexed into individual signals at the receiver.

A fiber link comprises of several fiber spans, where a fiber span is the WDM line segment between two consecutive EDFA amplifiers. Moreover, a fiber link has a single fiber in each direction. In Figure 2-1, only one direction of a fiber link is shown. There are add and drop ports at each node for data to locally enter and leave the network at transmitters (Tx) and receivers (Rx), respectively. At each node, an all-optical switching fabric, which can switch an optical signal from any input port to any other output port, demultiplexes and switches each incoming signal.

In order to facilitate regeneration, an optional pool of regenerators can be placed between transmitters and receivers as shown in Figure 2-1. For a lightpath, a transparent segment including one or more links between two consecutive nodes where regeneration takes place is referred to as a regeneration segment of the lightpath. Thus, a lightpath comprises of one or more regeneration segments. If a lightpath does not require any regeneration along its path, then we refer to it as a transparent lightpath. After a signal has been regenerated, it resumes the original characteristics it had at the source node.

2-2 Notations

In this section, we describe the notations used in the rest of this thesis.

- N is the number of nodes in the network.
- E represents the number of fiber links in the network.
- N_R is the number of regenerator nodes in the network.
- $G = (\mathcal{N}, \mathcal{L})$ is undirected graph representing the network topology with vertex set $\mathcal{N}$ (representing the network nodes), and arc set $\mathcal{L}$ (representing the network fiber links).
- (u, v) denotes the fiber link between nodes u and v that are adjacent.
- m_q is the number of QoS metrics associated with each fiber link.
- m_p is the number of the physical impairments associated with each fiber link.
- $\overrightarrow{q}(u, v)$ represents the QoS metrics vector associated with the fiber link (u, v), where $q_i(u, v)$, $1 \leq i \leq m_q$, is the i^{th} QoS metric on the fiber link.
- $\overrightarrow{\delta}(u, v)$ is the physical impairments vector associated with the fiber link (u, v), where $\delta_j(u, v)$, $1 \leq j \leq m_p$, is the j^{th} physical impairment on the fiber link.
- $\overrightarrow{T}$ denotes the QoS threshold vector, where T_i represents the threshold value of the i^{th} QoS metric $(1 \leq i \leq m_q)$.
- $\overrightarrow{\Delta}$ represents the physical impairment threshold vector, where Δ_j represents the threshold value of the j^{th} physical impairment $(1 \leq j \leq m_p)$.
- $\mathcal{R} = (s, t, \{T_i\}, \{\Delta_j\})$ denotes a lightpath request from the source node s to the destination node t subject to the constraints.
- $\overrightarrow{I}(p)$, is a vector of sums, where $I_i(p)$ represents the sum of the i^{th} physical impairment along the path p since the last regeneration (or since the source node if there was no regeneration).
- $\overrightarrow{I^*}(p)$, is a sum vector, where $I_i(p)$ denotes the sum of the i^{th} physical impairment along the path p since the last regenerator node (or since the source node if no regenerator node is encountered).
- $\overrightarrow{Q}(p)$, denotes the sum vector of the QoS metrics along the path p, where $Q_i(p)$ represents the sum for the i^{th} QoS metric.
- $Adj[u]$ represents the set of nodes that are adjacent to node u in the graph G.

- $\pi[u[i]]$ represents the set of nodes that appear in the i^{th} path stored at node u.
- $lur(u[i])$ represents the last unused (free) regenerator along the i^{th} path stored at node u.
- $reg\_numb(u[i])$ is the number of regenerators used in the i^{th} path stored at node u.
- $\mathcal{B}$ is a set of nodes including all the regenerator nodes in the network and the destination node of the request $\mathcal{R}$.
- $p^*_{n\to t;i}$ is the shortest path from a node n to the destination node t in terms of the QoS metric i.
- $p^{**}_{n\to j;i}$ represents the shortest path from a node n to the nearest node j, $j \in \mathcal{B}$, with respect to the i^{th} physical impairment.
- $\overrightarrow{b}(n)$ represents the attainable lower bounds for each QoS metric, with $b_i(n) = Q_i(p^*_{n\to t;i})$.
- $\overrightarrow{b^*}(n)$ denotes the attainable lower bounds on the distance to the nearest node j in $\mathcal{B}$ for each physical impairment, where $b^*_i(n)$ is the accumulated sum of the i^{th} physical impairment on the path $p^{**}_{n\to j;i}$.

2-3 Problem Definition

In this section, the impairment-aware QoS routing problem is defined. The physical optical network is modeled as an undirected graph $G = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of N nodes and $\mathcal{L}$ is the set of L links. Each fiber link $(u, v) \in \mathcal{L}$ is associated with a set of non-negative metrics, each representing a QoS metric $q_i(u, v)$ or a linear physical impairment $\delta_j(u, v)$ where $i \in \{1, 2, ..., m_q\}$ and $j \in \{1, 2, ..., m_p\}$. In addition, all the fiber links in the network are associated with the same wavelength λ. Each node has a given number of regenerators which could be zero. $\mathcal{N}_R \subseteq \mathcal{N}$ represents the set of R nodes that have regeneration capacity, and $N_R = |\mathcal{N}_R|$ represents the number of regenerator nodes. A lightpath request $\mathcal{R}$ is represented by the tuple $(s, t, \{T_i\}, \{\Delta_j\})$, where $s, t \in \mathcal{N}$ are the source and destination nodes of $\mathcal{R}$, $\{T_i\}$ and $\{\Delta_j\}$ are the set of threshold values for QoS and physical impairments, respectively. The impairment-aware QoS routing (IQR) problem is to find a simple path and allocate resources to a given request $\mathcal{R}$ such that the end-to-end QoS constraints are satisfied, and the physical impairments should not exceed their corresponding threshold values for any regeneration segment along the path.

A path involving loops cannot be a candidate for the solution of this problem because loops may violate the capacity of a link or a node. Hence, the problem requires a simple path solution. In the following section, we describe the general approach used in this thesis to solve the problem.

2-4 General Approach

In solving the IQR problem, we handle the QoS metrics and the physical impairments differently. This is due to the fact that a signal regains the physical attributes it had at the source node when it is regenerated at an intermediate node. Thus, the physical impairments accumulated along the sub-path before the regeneration should be reset to zero. In our approach, we avoid regeneration until the impairment level exceeds the acceptable threshold when using the outgoing link at the current node. For the physical impairments, the aforementioned vectors $\overrightarrow{I}(p)$ and $\overrightarrow{I^*}(p)$ are used to perform the regeneration process effectively. Whenever a regenerator node is encountered along a path p, $\overrightarrow{I}^*(p)$ is reset to $\overrightarrow{0}$. The need for regeneration arises when the accumulated physical impairment sum $I_i(p)$ exceeds Δ_i for at least one $i \in \{1, 2, ..., m_p\}$. Regeneration is possible only if there is an unused regenerator node along p since the last actual regeneration, or since the start node s if there was no actual regeneration. Whenever a regeneration is required and is possible, we do not need to recompute the path again starting from the last regenerator node. Rather, we perform regeneration in an effective way by simply copying $\overrightarrow{I^*}(p)$ to $\overrightarrow{I}(p)$. For the QoS metrics, the end-to-end sum vector is taken because they are not influenced by the regeneration of the signal.

Besides this general approach, both the exact and the heuristic algorithms employ their own specific approaches to solve the problem. The details of these algorithms are described in the following sections.

2-5 Exact Algorithm

In this section, the exact algorithms proposed to solve the two variants of the IQR problem are discussed. Our exact algorithms use some concepts of the self-adaptive multi-constrained routing algorithm (SAMCRA) suggested in [33]. SAMCRA is an exact multi-constrained shortest path algorithm, i.e, it certainly finds a path that optimally satisfies all the constraints, if it exists. Next, we give a brief description of SAMCRA before we proceed to explain how it can be modified to solve the IQR problem. The four major underlying concepts of SAMCRA are:

1. *A non-linear path length:* Different (sub)paths with multiple QoS metrics are compared using their path lengths. For a given path p, SAMCRA defines the path length $l(p)$ as a non linear function of the link weights (Equation 2-1).

$$l(p) = \max_{1 \le i \le m_q} \left[\frac{Q_i(p)}{T_i} \right] \tag{2-1}$$

2. *k-shortest paths:* At each intermediate node, SAMCRA stores up to k shortest paths (where k is not restricted) from the source to that node with their corresponding lengths.

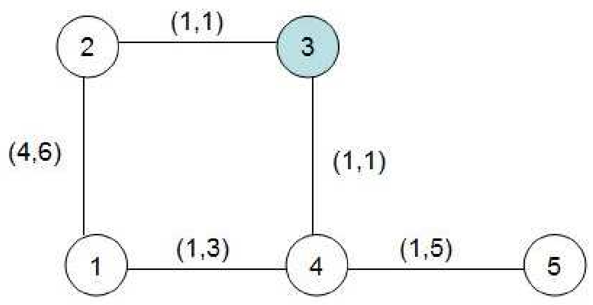


Figure 2-2: A typical example showing that the dominance concept fails for impairment-aware routing. The connection request is (1,5,7,7) and only node 3 has regeneration capacity.

3. *Path dominance:* A path is said to be dominated if it is higher or equal in every metric and exactly higher in at least one metric than another path. SAMCRA drops such dominated paths, thereby reducing the search space for possible paths.

4. *Look-ahead:* The look-ahead concept is an additional search space reduction mechanism, whereby the lower bounds related to the remaining (sub)path towards the destination are used in order to predict whether the current (sub)path can possibly exceed any of the QoS constraints. For each QoS metric, the lower bound is built by computing the shortest path tree rooted at the destination node to each node in the network.

The non-dominance technique of SAMCRA fails in impairment-aware routing due to regeneration. The network shown in Figure 2-2 is a typical example where the dominance concept fails. In this example, we are assuming that $m_q = m_p = 1$ and the request is $(1, 5, 7, 7)$. At node 2, the subpath $p_1 = 1 \to 2$ with $Q_1(p_1) = 4$ and $I_1(p_1) = I_1^*(p_1) = 6$ is dominated by the subpath $p_2 = 1 \to 4 \to 3 \to 2$ with $Q_1(p_2) = 3$ and $I_1(p_2) = I_1^*(p_2) = 5$. If we drop p_1, a simple path solution cannot be obtained for the current request. However, if we keep p_1, it will be part of the only feasible path $(1 \to 2 \to 3 \to 4 \to 5)$.

SAMCRA cannot be directly applied to solve the IQR problem because it does not have the capability to handle regeneration, and the non-dominance technique also fails due to regeneration. Therefore, we have built exact algorithms that use some concepts of SAMCRA and also incorporate additional features to handle regeneration. In our algorithms, we adopt the non-linear definition of path length and the k-shortest paths concept. For the physical impairments, we define the path length of a path p as:

$$l^*(p) = \max_{1 \leq i \leq m_p} \left[\frac{I_i(p)}{\Delta_i} \right] \tag{2-2}$$

For the QoS metrics, the look-ahead can be applied in the same way as SAMCRA. Let $p^*_{n \to t;i}$ be the shortest path from node n to the destination node t for QoS metric

i. Thus, the attainable lower bounds for each QoS metric is represented by the vector $\overrightarrow{b}(n)$ with $b_i(n) = Q_i(p^*_{n\rightarrow t;i})$. At each node n, SAMCRA stores the length of the sum of the accumulated sum vector and the lower bound vector, i.e, $l(\overrightarrow{Q}(p_{s\rightarrow n}) + \overrightarrow{b}(n))$. This "predicted" length of the path is used as the comparison metric to extract the minimum length path from the queue of the stored paths. As a result, the paths with the lowest "predicted" end-to-end length are given priority than the path with the lowest length so far. Further, SAMCRA uses a parameter *maxlength* representing the maximum length that a (sub)path may have. If the length of a (sub)path exceeds *maxlength*, it either means that the constraints are violated or a shorter end-to-end path is previously found. Thus, a (sub)path with length greater than *maxlength* can be discarded.

For the physical impairments, the look-ahead can be applied with some modifications. At each node n, we compute the look-ahead sum $I^*_i(p_{s\rightarrow n}) + b^*_i(n)$ for each $i = 1, 2, ..., m_p$. This sum represents the sum of the i^{th} physical impairment between the last regenerator node and the next nearest regenerator node or the destination node. If the sum exceeds the threshold value Δ_i, then the (sub)path is discarded.

In SAMCRA, there is no need of explicit checking for occurrence of loops, because the non-dominance test automatically avoids loops. As a result of the failure of the non-dominance technique, we need to explicitly check for loops in impairment-aware QoS routing.

In the following subsections, the exact algorithms for the two minimization objectives, namely, *QoS optimization* and *regenerator-count minimization*, are presented.

2-5-1 QoS Optimization

In this section, we present the exact algorithm used to solve the IQR problem aiming at minimizing the end-to-end QoS path length defined in Equation 2-1. Since we tend to optimally satisfy the QoS constraints, we refer to this problem as the Impairment-aware Optimal QoS Routing (IOQR) problem. Similarly, the algorithm is referred to as Exact Impairment-aware Optimal QoS Routing Algorithm (EIOQRA). EIOQRA constantly computes paths and stores them in a queue and then extracts the path with the minimum measure from the queue. Then, the extracted path is extended to the neighboring nodes. If an extended path is eligible to be stored in the queue, then it is inserted into the queue; otherwise it is discarded. This process of insertion into and extraction from the queue continues until either the queue is empty or a path at the destination node is extracted. If the queue is empty, then there is no path satisfying the given QoS constraints. If a path at the destination node is extracted, then the path is the optimal solution.

A path is stored in the queue along with its QoS predicted length, its regenerator-count, and its length in terms of the physical impairments. The paths are extracted from the queue according to the following rules:

- The path with the minimum predicted length (with respect to the QoS metrics) is extracted, and
- If several paths have the same length, the one with the minimum regenerator count is extracted, and
- If they have the same regenerator count, the one with the lowest length with respect to the physical impairments is extracted.

A (sub)path P will be discarded:

- If its predicted QoS length exceeds the maximum length of the QoS metrics, or
- If at a given node, any of the physical impairments exceeds its acceptable threshold even after regeneration (if possible), or
- If the look-ahead sum of any of the physical impairments exceeds its threshold value.

The metacode of the algorithm is described in the next subsection.

The Meta Code

In this section, we describe the meta-code of the exact algorithm aiming at QoS optimization. In addition to the notations described in Section 2-2, the following parameters are used in the algorithm. The parameters $maxlen\_q$ and $maxlen\_p$ refer to the maximum lengths of the QoS measure and physical impairment, respectively. For the QoS metrics, we drop a (sub)path p with length $l(p) > maxlen\_q$ because it either violates the constraints or is longer than an already found end-to-end path. Whereas, for the physical impairment, if $l^*(p) > maxlen\_p$, we try to regenerate the signal; and if regeneration is not possible, the (sub)path p is discarded. The number of paths that are stored at a node n is represented by $counter[n]$. The parameter $predicted\_length$ denotes the predicted end-to-end length of a path in terms of the QoS metrics, whereas, $look\_ahead\_p$ represents the length from the last regenerator node to the nearest regenerator node/destination node with respect to the physical impairments. The parameters reg_count and $last_reg$ are used to temporarily store the regenerator-count and the last unused regenerator, respectively, of the extended path. The queue where the paths are stored is denoted by Z.

In the INITIALIZE subroutine listed in Algorithm 1, the necessary parameters for the main algorithm are initialized and the look-ahead information is computed. Lines 1 and 2 initialize $counter$ of each node to zero. In lines 4 and 5, $maxlen\_q$ and $maxlen\_p$ are initialized to 1.0.

Algorithm 1 INITIALIZE(G, m_p, m_q, s, t)

```
1: for each $v \in \mathcal{N}$ do {* initialize the stored path counter of each node $v$ to 0 * }
2:     counter$[v] \leftarrow 0$
3: end for
4: maxlen_q $\leftarrow$ 1.0          {* initialize the maximum QoS length to 1.0 *}
5: maxlen_p $\leftarrow$ 1.0     {* initialize the maximum physical impairment length to 1.0 *}
6: for $i = 1, ..., m_q$ do
7:     DIJKSTRA$(G, t, i) \rightarrow b_i(n)$    { * compute the lower bound $b_i(n)$ for each node $n$ *}
8: end for
9: for $i = 1, ..., m_p$ do
10:    DIJKSTRA$(G, \mathcal{B}, i) \rightarrow b_i^*(n)$     {* compute the lower bound $b_i^*(n)$ for each node $n$ *}
11: end for
12: queue $Z \leftarrow \emptyset$                         {* initialize $Z$ to an empty queue *}
13: counter$[s]$ $\leftarrow$ counter$[s]$ + 1          { * increment the counter of the source node $s$ *}
14: $last\_reg \leftarrow 0$           {* $last\_reg$ is the last unused regenerator and is initialized to 0 *}
15: INSERT $(Z, s,$counter$[s]$,NIL,$l(\overrightarrow{b}(s)), 0, 0, last\_reg, 0)$      {* insert $s$ into the queue *}
```

The DIJKSTRA function in line 7 computes the look-ahead lower bounds $\overrightarrow{b}$ for the QoS metrics. Here, for each individual QoS measure i, the lower bounds $b_i(n)$ from any node $n \in \mathcal{N}$ to the destination node t is calculated. The lower bounds are found efficiently by computing, for each QoS measure, a shortest path tree with the Dijkstra's algorithm from the destination to all other nodes. Then the shortest distance $b_i^*(n)$ from each node $n \in \mathcal{N}$ to the nearest node in $\mathcal{B}$ is calculated for each individual physical impairment i (line 10). This is done by computing, for each physical impairment, the shortest path tree with the Djikstra's algorithm from each node in $\mathcal{B}$ to all other nodes. In line 14 the last unused regenerator ($last\_reg$) is set to 0. Finally, INITIALIZE inserts the source node s into the queue Z (line 15).

The main algorithm (EIOQRA) is listed in Algorithm 2. The algorithm starts by executing the subroutine INITIALIZE in line 1. Then EXTRACT_MIN function extracts the minimum path length in the queue according to the extraction rule described before as long as the queue is not empty (lines 2-3). But if the queue Z is empty, then there is no feasible path and the algorithm terminates. The extracted path $u[i]$ represents the i^{th} path $p_{s \rightarrow u}$ stored in the queue at node u. In line 4, the extracted path is marked gray. If the node u corresponding to the extracted path $u[i]$ equals the destination t, then $u[i]$ is the shortest path satisfying the constraints and the algorithm stops by returning $u[i]$ (line 6). But, if $u \neq t$, the neighbors of u are scanned starting from line 8.

In lines 8-42, the i^{th} path up to node u is extended toward its neighboring node v, except for the previous nodes in the path $u[i]$. In order to prevent looping, line 8 explicitly checks by back tracing if v is in the previous nodes of $u[i]$. If node v passes the looping

Algorithm 2 EIOQRA(G, m_p, m_q, s, t)

```
1: INITIALIZE$(G, m, s, t) \rightarrow \overrightarrow{b}, \overrightarrow{b^*}$
2: while $(Z \neq \emptyset)$ do {* if the queue $Z$ is not empty, extract a path *}
3:   EXTRACT-MIN$(Z) \rightarrow u[i]$   {* extract the minimum-length path *}
4:   $u[i] \leftarrow$ GREY
5:   if $(u = t)$ then {* if a path at the destination node is extracted *}
6:     STOP$\rightarrow$ return path
7:   else
8:     for each $v \in$Adj$[u]$ AND $v \notin \{\pi[u[i]], s\}$ do {* extend the path to each adjacent node which is not in the previous nodes along the path *}
9:       $flag \leftarrow 0$   {* $flag$ is used to track regeneration *}
10:      $\overrightarrow{\alpha} \leftarrow \overrightarrow{I^*}(u[i]) + \overrightarrow{\delta}(u \rightarrow v)$   {* compute $\overrightarrow{\alpha} = \overrightarrow{I^*}$(the extended path) *}
11:      $\overrightarrow{\beta} \leftarrow \overrightarrow{I}(u[i]) + \overrightarrow{\delta}(u \rightarrow v)$   {* compute $\overrightarrow{\beta} = \overrightarrow{I}$(the extended path) *}
12:      if $(l(\overrightarrow{\beta}) > maxlen\_p)$ then {* if the threshold is exceeded *}
13:        if (lur$(u[i]) \neq 0$) then {* if there is a free regenerator *}
14:          $\overrightarrow{\beta} \leftarrow \overrightarrow{\alpha}$   {* perform regeneration *}
15:          $flag \leftarrow 1$   {* remember the regeneration by setting $flag = 1$ *}
16:        end if
17:      end if
18:      if $(flag = 1)$ then {* if regeneration has just taken place *}
19:        $reg\_count \leftarrow reg\_numb(u[i]) + 1$   {* increment the regenerator-count *}
20:        $last\_reg \leftarrow 0$   {* set the last unused regenerator to 0 *}
21:      else
22:        $reg\_count \leftarrow reg\_numb(u[i])$   {* copy the regenerator-count from $u[i]$ *}
23:        $last\_reg \leftarrow lur(u[i])$   {* copy the last unused regenerator from $u[i]$ *}
24:      end if
25:      predicted_length$\leftarrow l(\overrightarrow{Q}(u[i]) + \overrightarrow{q}(u \rightarrow v) + \overrightarrow{b}[v])$  {* compute the QoS predicted length *}
26:      look_ahead_p$\leftarrow l(\overrightarrow{I^*}(u[i]) + \overrightarrow{\delta}(u \rightarrow v) + \overrightarrow{b^*}[v])$   {* compute the physical impairment length of the path from the last regenerator to the nearest regenerator *}
27:      if $((precicted\_length \leq maxlen\_q$ AND $look\_ahead\_p \leq maxlen\_p$ AND $(l(\overrightarrow{\beta}) \leq maxlen\_p))$ then {* if the extended path is eligible to be stored *}
28:        if $reg(v) \neq 0$ then {* if the current node is a regenerator node *}
29:          $last\_reg \leftarrow v$   {* set the last unused regenerator to $v$ *}
30:          $\overrightarrow{\alpha} \leftarrow \overrightarrow{0}$   {* reset $\overrightarrow{\alpha}$ to $\overrightarrow{0}$ *}
31:        end if
32:        counter$[v]\leftarrow$ counter$[v]$+1   {* increment the counter of node $v$ *}
33:        INSERT$(Z, v,$counter$[v]$,$predicted\_length, l(\overrightarrow{\beta}), reg\_count, last\_reg, l(\overrightarrow{\alpha}))$   {* insert the extended path into the queue and save its necessary parameters *}
34:        $\overrightarrow{Q}(v[$counter$[v]]) \leftarrow \overrightarrow{Q}(u[i]) + \overrightarrow{q}(u \rightarrow v)$
35:        $\overrightarrow{I}(v[counter[v]]) \leftarrow \overrightarrow{\beta}$
36:        $lur(v[counter[v]]) \leftarrow last\_reg$
37:        $\overrightarrow{I^*}(v[counter[v]]) \leftarrow \overrightarrow{\alpha}$
38:        $\pi[v[$counter$[v]]] \leftarrow u[i]$
39:        if $(v = t$ AND $predicted\_length < maxlen\_q)$ then {* update $maxlen\_q$ *}
40:          maxlen_q $\leftarrow$ predicted_length
41:        end if
42:      end if
43:    end for
44:  end if
45: end while
```

test, then the physical impairment weight vector ($\overrightarrow{\beta}$), and the weight vector of the physical impairment since the last regenerator ($\overrightarrow{\alpha}$) of the extended path are computed (lines 10-11). Line 12 tests if the physical impairment weight of the extended path violates the physical constraint. If it does violate, the availability of an unused regenerator along the subpath $u[i]$ is checked (line 13); and if one is found regeneration takes place at the regenerator node. At regeneration, the weight vector of the physical impairment is set to the weight vector of the physical impairment beginning from the last unused regenerator (line 14). In line 15, the $flag$ is set to 1 to remember that a regenerator has just been used. In lines 18 to 24, the values of regenerator count (*reg_count*) and the last unused regenerator (*last_reg*) of the extended path are updated depending on the value of $flag$. If a regenerator has been used ($flag = 1$), *reg_count* is incremented, and *last_reg* is set to 0. Otherwise the corresponding values are copied from $u[i]$.

The length of the predicted end-to-end QoS weight vector is calculated in line 25. The predicted end-to-end QoS weight vector comprises of the actual subpath weight vector from s to v plus the lower bound vector from v to t. The *look_ahead_p* calculated in line 26 refers to the length of the predicted physical impairment vector that is composed of the actual subpath weight vector from s to v plus the lower bound vector from v to the nearest node $n \in \mathcal{B}$. Line 27 checks if the new extended path qualifies to be stored. If the path does not qualify, then it is discarded and the scanning continues in line 8 with the next neighbor of u. Otherwise, the commands in lines 28-42 are executed. If v has an unused regenerator, *last_reg* and $\overrightarrow{\alpha}$ are updated accordingly (lines 28 to 29). In lines 32-33, the *counter* of node v is incremented, and the new extended path is inserted into the queue Z. In lines 34-38, the new extended path is stored at the current *counter* index of v with all its parameters. If v is the destination node t, and if its end-to-end predicted QoS length is less than *maxlen_q*, then *maxlen_q* is updated in lines 39 and 40. The paths for which *predicted_length* > *maxlen_q* are dropped to reduce the search space.

Complexity of EIOQRA

In this section we present the worst-case complexity of EIOQRA. First, we compute the complexity of the INITIALIZE subroutine as follows. It takes O(N) times to initialize the counter. Executing the heap-optimized Dijkstra algorithm in line 6-8 takes $m_q O(NlogN + E)$. Further, in line 10, the Dijkstra's algorithm is executed N_R times, leading to a time complexity of $m_p O(N_R(NlogN + E))$. All the other operations take $O(1)$. Thus, the overall worst-case complexity of the initialization phase is $O((m_p + m_q)(N_R NlogN + N_R E))$.

The overall complexity of the main algorithm (EIOQRA) is computed as follows. The initialization phase in line 1 takes $O((m_p + m_q)N_R NlogN + (m_p + m_q)N_R E)$. The maximum number of paths that the queue Z can contain is $k_{max}N$. Selecting the minimum path length among $k_{max}N$ different path lengths from a queue that is structured with

Fibonacci or relaxed heap takes at most $O(log(k_{max}N))$ [33]. Since k_{max} paths can be stored at each node in the queue, the EXTRACT_MIN function in line 3 at most takes $O(k_{max}Nlog(k_{max}N))$. Returning the path in line 6 takes at most $O(N)$. Since the for loop starting in line 8 is invoked at most k_{max} times from each side of each link in the graph, it takes $O(k_{max}E)$ time at most. The test conducted in line 8 to avoid loops takes $O(N)$. The operations in lines 9-32 and 33-38 take $O(m_q + m_p)$. The insertion into the queue in line 33 can be done in $O(1)$. Therefore, the overall complexity of EIOQRA becomes $O((m_p + m_q)N_R NlogN + (m_p + m_q)N_R E) + k_{max}Nlog(k_{max}N) + k_{max}E(N + (m_q + m_p)))$. If we assume $m_q = m_p = m$, then complexity of EIOQRA simplifies to:

$$C_{EIOQRA} = O(mN_R NlogN + mN_R E + k_{max}Nlog(k_{max}N) + k_{max}EN + mk_{max}E) \qquad (2\text{-}3)$$

For any network, $k_{max} \leq \lfloor e(N-2)! \rfloor$, where $e \simeq 2.718$ [33].

An Example Network

Figure 2-3(a) shows an example network, where only nodes 3 and 4 have regeneration capacity. There are two QoS constraints with $\overrightarrow{T} = (20, 20)$ and two physical impairments with $\overrightarrow{\Delta} = (10, 10)$. On each link are shown four numbers, the first two on the left side of the vertical line segments represent the QoS metrics and the other two on the right side are metrics representing the physical impairments associated with the link. In this example, we try to find a path that satisfies the request (1, 7, (20, 20), (10, 10)).

Initialization phase (Figure 2-3(b)): First, $b_i(n)$ and $b_i^*(n)$ will be computed using Dijkstra's algorithm. Two vectors are shown in rectangular boxes as shown in Figure 2-3(b), where the vector to the left and right of the vertical segments represent $\overrightarrow{b}$ and $\overrightarrow{b^*}$, respectively. Further, *maxlen_q* and *maxlen_p* are both initialized to 1. With this information, we start the actual operation of EIOQRA as depicted in Figure 2-4(a).

Step 1 (Figure 2-4(a)): We begin by scanning the neighbors of the source node. Its neighbors are node 2 and node 3. The path to node 2 has a predicted length $l(p^1_{1\to 2}) = l(\overrightarrow{Q}(p^1_{1\to 2}) + \overrightarrow{b}(2))$= $\max(\frac{1+5}{20}, \frac{5+7}{20})$= 0.6, a regenerator count $reg\_numb(p^1_{1\to 2}) = 0$, the length of the physical impairment vector $l^*(p^1_{1\to 2})$=$\max(\frac{2}{10}, \frac{3}{10})$= 0.3, and shortest impairment distance to the nearest regenerator node (or to t) $look\_ahead\_p(p^1_{1\to 2}) = max(\frac{2+7}{10}, \frac{3+4}{10}) = 0.9$. The path to node 3 has $l(p^1_{1\to 3})$=0.65, $reg\_numb(p^1_{1\to 3}) = 0$, $l^*(p^1_{1\to 3})$=0.8, and $look\_ahead\_p(p^1_{1\to 3}) = 0.8$. For both paths, the predicted lengths are less than *maxlen_q*, the length of the physical impairment vectors and the *look_ahead_p* values are less than *maxlen_p*. Thus, we store both of them at their corresponding nodes.

Step 2 (Figure 2-4(b)): Since node 2 has the minimum length subpath ($l(p^1_{1\to 2}) = 0.6$) stored so far in the queue, it is extracted next. Its neighbors are nodes 1 and

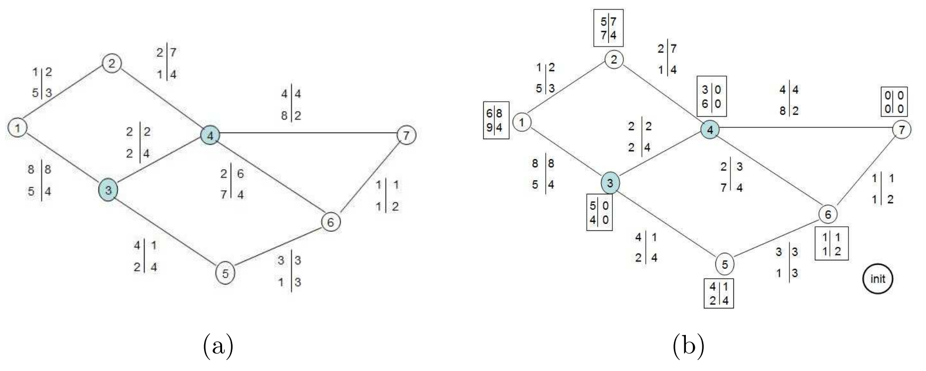


Figure 2-3: (a) An example network, (b) Initialization step

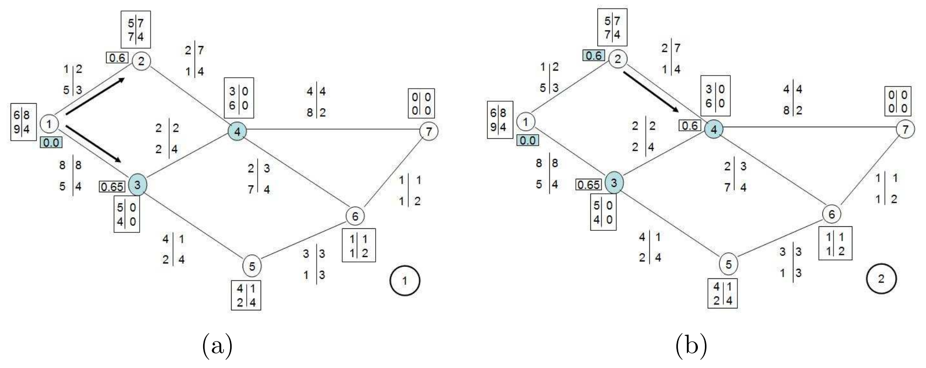


Figure 2-4: (a) Step 1, (b) Step 2

4. But node 1 is the previous node, so it should not be scanned. For node 4, path $p^1_{1\to4} = 1 \to 2 \to 4$, with $l(p^1_{1\to4})$=max$(\frac{1+2+3}{20}, \frac{5+1+6}{20})$=0.6, $reg\_numb(p^1_{1\to4})$=0, $l^*(p^1_{1\to4})$=0.9, and $look\_ahead\_p(p^1_{1\to4}) = max(\frac{2+7+0}{10}, \frac{3+4+0}{10}) = 0.9$ is added to the queue.

Step 3 (Figure 2-5(a)): Node 4, which is the minimum length node ($l(p^1_{1\to4}) = 0.6$), is extracted from the queue. All its neighbors, except node 2, which is the previous node, are scanned. For node 3, we find a second path, $p^2_{1\to3} = 1 \to 2 \to 4 \to 3$, with predicted length $l(p^2_{1\to3}) = 0.6$, $reg\_numb(p^2_{1\to3})$=0, $l^*(p^2_{1\to3}) = 1.1$, and $look\_ahead\_p(p^2_{1\to3}) = 0.4$. However, since $l^*(p^2_{1\to3}) = 1.1 > maxlen\_p$, a regeneration at node 4 is required. After regeneration, the history regarding the physical impairment is reset. Thus, $reg\_numb(p^2_{1\to3}) = 1$, and $l^*(p^2_{1\to3}) = 0.4$. Similarly, for nodes 6 and 7 regeneration is required for their respective subpaths at node 4. After regeneration, the path at node 6 is $p^1_{1\to6} = 1 \to 2 \to 4 \to 6$ with $l(p^1_{1\to6}) = 0.7$, $reg\_numb(p^1_{1\to6}) = 1$, $l^*(p^1_{1\to6}) = 0.4$, and $look\_ahead\_p(p^1_{1\to6}) = 0.4$; whereas the path at node 7 is $p^1_{1\to7} = 1 \to 2 \to 4 \to 7$, with $l(p^1_{1\to7}) = 0.7$, $reg\_numb(p^1_{1\to7}) = 1$, $l^*(p^1_{1\to7}) = 0.4$, and $look\_ahead\_p(p^1_{1\to7}) = 0.4$. Since node 7 is the destination node, and since $l(p^1_{1\to7} = 0.7 < maxlen\_q)$, $maxlen\_q$ is lowered to 0.7.

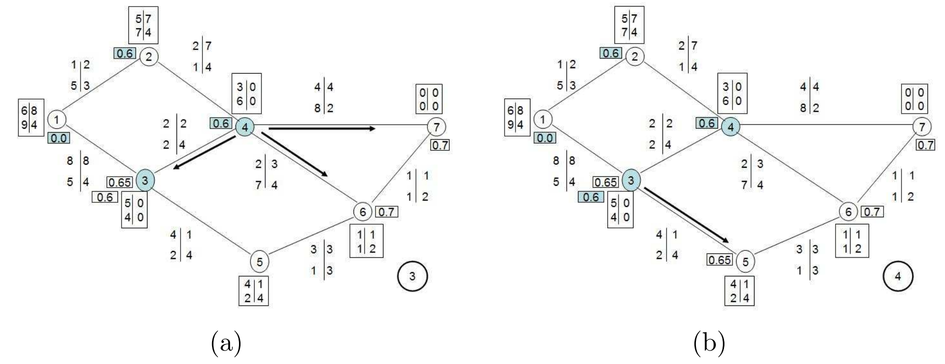


Figure 2-5: (a) Step 3, (b) Step 4

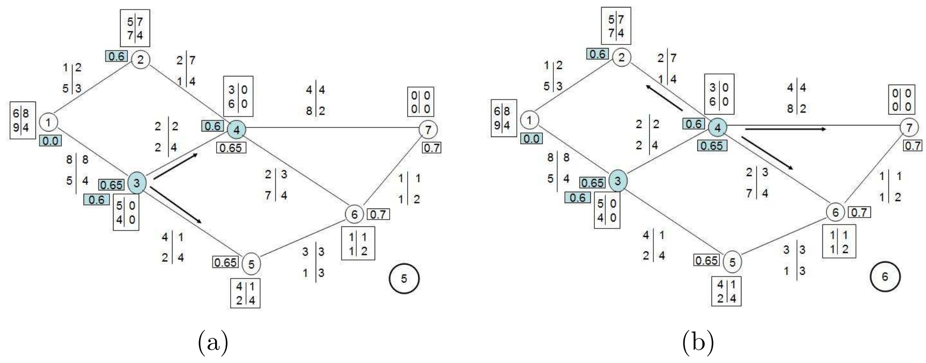


Figure 2-6: (a) Step 5, (b) Step 6

Step 4 (Figure 2-5(b)): Node 3, which has the minimum length subpath ($l(p^2_{1\to3}) = 0.6$), is extracted. From node 3, we obtain a new subpath to node 5, $p^1_{1\to5} = 1 \to 2 \to 4 \to 7$, with $l(p^1_{1\to5}) = 0.65$, $reg\_numb(p^1_{1\to7}) = 1$, $l^*(p^1_{1\to5}) = 0.8$, and $look\_ahead\_p(p^1_{1\to5}) = 0.8$. This path is stored in the queue because it satisfies all the requirements.

Step 5 (Figure 2-6(a)): In this step, we have two subpaths in the queue, $p^1_{1\to3}$ and $p^1_{1\to5}$, which have the same predicted length 0.65. However, since $reg\_numb(p^1_{1\to3}) < reg\_numb(p^1_{1\to5})$, node 3 is extracted. The path $p^2_{1\to4} = 1 \to 3 \to 4$ towards node 4 with predicted length $l(p^2_{1\to4}) = 0.65$, $reg\_numb(p^2_{1\to4}) = 0$, $l^*(p^2_{1\to4}) = 1$, and $look\_ahead\_p(p^2_{1\to4}) = 0.4$ is stored. However, the path towards node 5 is not stored because the predicated length $l(P^2_{1\to5}) = max(\frac{8+4+4}{20}, \frac{5+2+2}{20}) = 0.8$ is larger than $maxlen\_q$ $= 0.7$. *Step 9 (Figure 2-8):* Finally, the shortest path at node 7, which is the destination node, is extracted. Hence the path $p^2_{1\to7} = 1 \to 3 \to 5 \to 6 \to 7$ is returned, thereby terminating the algorithm.

Step 6 (Figure 2-6(b)): In this step, we have two subpaths in the queue, $P^2_{1\to4}$ and $P^1_{1\to5}$, which have the same predicted length 0.65. However, since $reg\_numb(P^2_{1\to4}) < reg\_numb(P^1_{1\to5})$, node 4 is extracted. Nodes 2, 6, and 7 are

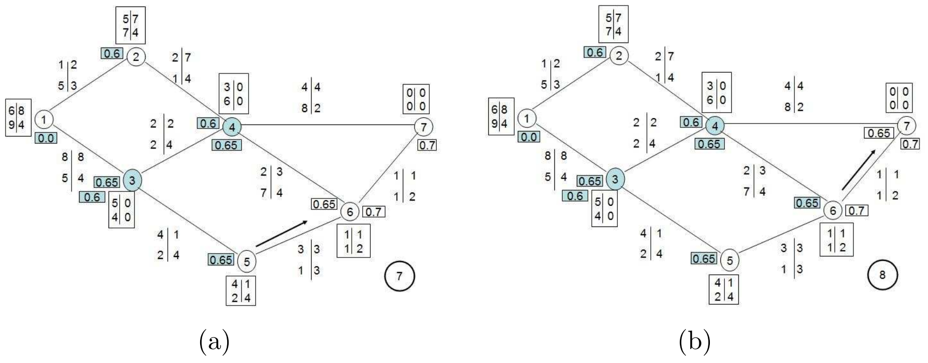


Figure 2-7: (a) Step 7, (b) Step 8

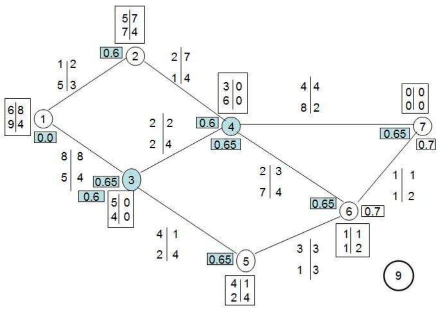


Figure 2-8: Step 9

scanned. The path to node 2, $p^2_{1\to2} = 1 \to 3 \to 4 \to 2$, is not stored because $look\_ahead\_p(p^2_{1\to2}) = 1.4 > maxlen\_p$. The new path to node 6, $p^2_{1\to6} = 1 \to 3 \to 4 \to 6$, is not stored either because its predicted length $l(p^2_{1\to6}) = 0.75 > maxlen\_q$. Further, the new path to node 7 is discarded because $l(p^2_{1\to7}) = 0.75 > maxlen\_q$.

Step 7 (Figure 2-7(a)): In this step, the first path at node 5, $p^1_{1\to5}$ with $l(p^1_{1\to5}) = 0.65$, has the minimum predicted length and is extracted from the queue. Here we find a second path at node 6 with, $l(p^2_{1\to6}) = 0.65$, $reg\_numb(p^2_{1\to6}) = 1$, $l(\overrightarrow{I}(p^2_{1\to6})) = 1.1$, and $look\_ahead\_p(p^2_{1\to6}) = 0.9$. However, since $l^*(p^2_{1\to6}) = 1.1 > maxlen\_p$, a regeneration at node 3 is required. After regeneration, the history regarding the physical impairment is reset. Thus, $reg\_numb(p^2_{1\to6}) = 2$, and $\overrightarrow{I}(p^2_{1\to6}) = (4, 7)$.

Step 8 (Figure 2-7(b)): Since node 6 has the minimum length subpath ($l(P^2_{1\to6}) = 0.65$) stored so far in the queue, it is extracted next. Its neighbor 7 is scanned, but 4 and 5 are not because they are in the predecessor list of $P^2_{1\to6}$. The second path to node 7, $P^2_{1\to7}$, with $l(P^2_{1\to6}) = 0.65$, $reg\_numb(P^2_{1\to7}) = 2$, $l^*(P^2_{1\to7}) = 0.9$, and $look\_ahead\_p(P^2_{1\to7})) = 0.9$ is stored.

2-5-2 Regenerator-Count Minimization

In this section, we provide the exact algorithm for the IQR problem with the objective of minimizing the regenerator-count. The impairment-aware QoS routing with regenerator minimization (IQRRM) problem is to find a simple path and allocate resources to a given request such that the end-to-end QoS requirements are satisfied, the physical impairments of each regenerator segment do not exceed their corresponding threshold values, and the number of regenerators used is minimized.

Our exact algorithm is referred to as the Exact Impairment-aware QoS Routing with Regenerator Minimization (EIQRRM). The operation process of EIQRRM is similar to that of EIOQRA (the QoS optimization exact algorithm). The difference of the two algorithms lies in their path extraction rules and the criteria they use to discard a path. Instead of the *maxlen_q* parameter used in EIOQRA, EIQRRM uses the parameter *max_reg_count* to reduce the search space of the possible paths. *max_reg_count* stores the regenerator-count of the best feasible end-to-end path found so far. Thus, if the regenerator-count of a path is greater or equal to *max_reg_count*, it is discarded.

In EIQRRM, the paths are extracted from the queue according to the following rules:

- The path with the minimum regenerator-count is extracted, and
- If several paths have the same regenerator count, the path with the minimum predicted length (with respect to the QoS metrics) is extracted, and
- If they have the same predicted length, the one with the lowest length with respect to the physical impairments is extracted.

And we discard a (sub)path P if:

- If its regenerator-count is greater or equal to *max_reg_count*, or
- If its predicted QoS length exceeds *maxlen_q*, or
- If at a given node, any of the physical impairments exceeds its acceptable threshold even after regeneration (if possible), or
- If the look-ahead sum of any of the physical impairments exceeds its threshold value.

The details of the algorithm are described in the next subsection.

Algorithm 3 EIQRRM(G, m_p, m_q, s, t)

```
1: INITIALIZE$(G, m, s, t) \rightarrow \overrightarrow{b}, \overrightarrow{b^*}$
2: max_reg_count $\leftarrow N_R + 1$ {* initialize the maximum regenerator-count to $N_R + 1$ *}
3: while $(Z \neq \emptyset)$ do {* if the queue is not empty, extract a path *}
4:   EXTRACT-MIN$(Z) \rightarrow u[i]$   {* extract the minimum-length path *}
5:   $u[i] \leftarrow$ GREY
6:   if $(u = t)$ then {* if a path at the destination node is extracted *}
7:     STOP$\rightarrow$ return path
8:   else
9:     for each $v \in$Adj$[u]$ AND $v \notin \{\pi[u[i]], s\}$ do {* extend the path to adjacent nodes *}
10:      $flag \leftarrow 0$   {* $flag$ is used to track regeneration *}
11:      $\overrightarrow{\alpha} \leftarrow \overrightarrow{I^*}(u[i]) + \overrightarrow{\delta}(u, v)$   {* compute $\overrightarrow{\alpha} = \overrightarrow{I^*}$(the extended path) *}
12:      $\overrightarrow{\beta} \leftarrow \overrightarrow{I}(u[i]) + \overrightarrow{\delta}(u, v)$   {* compute $\overrightarrow{\beta} = \overrightarrow{I}$(the extended path) *}
13:      if $(l(\overrightarrow{\beta}) > maxlen\_p)$ then {* if the threshold is exceeded *}
14:        if $(\text{lur}(u[i]) \neq 0)$ then {* if there is a free regenerator *}
15:          $\overrightarrow{\beta} \leftarrow \overrightarrow{\alpha}$   {* perform regeneration *}
16:          $flag \leftarrow 1$   {* remember the regeneration by setting $flag = 1$ *}
17:        end if
18:      end if
19:      if $(flag = 1)$ then {* if regeneration has just taken place *}
20:        $reg\_count \leftarrow reg\_numb(u[i]) + 1$   {* increment the regenerator-count *}
21:        $last\_reg \leftarrow 0$   {* set the last unused regenerator to 0 *}
22:      else
23:        $reg\_count \leftarrow reg\_numb(u[i])$   {* copy the regenerator-count from $u[i]$ *}
24:        $last\_reg \leftarrow lur(u[i])$   {* copy the last unused regenerator from $u[i]$ *}
25:      end if
26:      predicted_length$\leftarrow l(\overrightarrow{Q}(u[i]) + \overrightarrow{q}(u, v) + \overrightarrow{b}[v])$ {* compute the QoS predicted length *}
27:      look_ahead_p$\leftarrow l(\overrightarrow{I^*}(u[i]) + \overrightarrow{\delta}(u, v) + \overrightarrow{b^*}[v])$   {* compute the physical impairment length of
         the path from the last regenerator to the nearest regenerator *}
28:      if $((precicted\_length \leq maxlen\_q$ AND $look\_ahead\_p \leq maxlen\_p$ AND $(l(\overrightarrow{\beta}) \leq maxlen\_p))$
         AND $reg\_count < max\_reg\_count$ then {* if the extended path is eligible to be stored *}
29:        if $reg(v) \neq 0$ then {* if the current node is a regenerator node *}
30:          $last\_reg \leftarrow v$   {* set the last unused regenerator to $v$ *}
31:          $\overrightarrow{\alpha} \leftarrow \overrightarrow{0}$   {* reset $\overrightarrow{\alpha}$ to $\overrightarrow{0}$ *}
32:        end if
33:        counter$[v]\leftarrow$ counter$[v]$+1   {* increment the counter of node $v$ *}
34:        INSERT$(Z, v,$counter$[v]$,$predicted\_length, l(\overrightarrow{\beta}), reg\_count, last\_reg, l(\overrightarrow{\alpha}))$   {* insert
           the extended path into the queue and save its necessary parameters *}
35:        $\overrightarrow{Q}(v[\text{counter}[v]]) \leftarrow (\overrightarrow{Q}(u[i]) + \overrightarrow{q}(u, v))$
36:        $\overrightarrow{I}(v[counter[v]]) \leftarrow \overrightarrow{\beta}$
37:        $lur(v[counter[v]]) \leftarrow last\_reg$
38:        $\overrightarrow{I^*}(v[counter[v]]) \leftarrow \overrightarrow{\alpha}$
39:        $\pi[v[\text{counter}[v]]] \leftarrow u[i]$
40:        if $(v = B$ AND $reg\_count < max\_reg\_count)$ then {* update $max\_reg\_count$ *}
41:          max_reg_count $\leftarrow$ reg_count
42:        end if
43:      end if
44:    end for
45:  end if
46: end while
```

The Meta Code

In this section, we describe the meta-code of the exact algorithm to solve the IQRRM problem. The main algorithm (EIQRRM) is listed in Algorithm 3. Because of the similarity of the problems they solve, EIQRRM and EIOQRA, which is listed in Algorithm 2 (Section 2-5-1), share a lot in common. EIQRRM differs from EIOQRA in the following aspects. First, EIQRRM uses an additional parameter *max_reg_count* which is initialized in line 2. We initialize *max_reg_count* to the total number of regenerators in the network N_R, and it will be updated dynamically to the regenerator-count of the best end-to-end path found.

Secondly, the EXTRACT_MIN function in line 4 of EIQRRM differs from that of EIOQRA in that it extracts the path based on the extraction rules described in Section 2-5-2. Thirdly, in line 28, EIQRRM adds another condition the extended path should meet to be stored in the queue. In addition to satisfying the look-ahead conditions and the physical constraint, the extended path should also have a regenerator count smaller than *max_reg_count*. For a given (sub)path, if the regenerator-count $reg\_count \geq max\_reg\_count$, then it means that a feasible ene-to-end path with regenerator-count equal to *max_reg_count* is already found. In that case, the subpath is dropped thereby reducing the search space. In lines 40 and 41, *max_reg_count* is updated. The remaining lines of EIQRRM are taken from EIOQRA. Moreover, the two algorithms have the same complexity.

An Example Network

An example network is shown in Figure 2-9(a), where only nodes 3 and 4 have regeneration capacity. In this example, we assume that $m_q = m_p = 2$ with the corresponding constraint vectors $\overrightarrow{T}$ = (20, 20) and $\overrightarrow{\Delta}$ = (10,10). The link metrics are represented by the four numbers shown on each link. The first two on the left side of the vertical line segments represent the QoS metrics and the other two on the right side are metrics representing the physical impairments associated with the link. Our goal in this example is to find a path that satisfies the request $(1, 7, (20, 20), (10, 10))$ with the minimum regenerator count.

Initialization phase (Figure 2-9(b)): First, Dijkstra's algorithm is used to compute $b_i(n)$ and $b_i^*(n)$. The two vectors are shown in rectangular boxes in Figure 2-9(b), where $\overrightarrow{b}$ and $\overrightarrow{b^*}$ are represented by the vector to the left and right of the vertical segments, respectively. *maxlen_q* and *maxlen_p* are also initialized to 1. In addition, *max_reg_count* is initialized to 3. We start the actual operation of EIQRRM with this information as shown in Figure 2-10(a).

Step 1 (Figure 2-10(a)): The source node (node 1) is extracted from the queue, and its neighbors (nodes 2 and 3) are scanned. The path to node 2 has a pre-

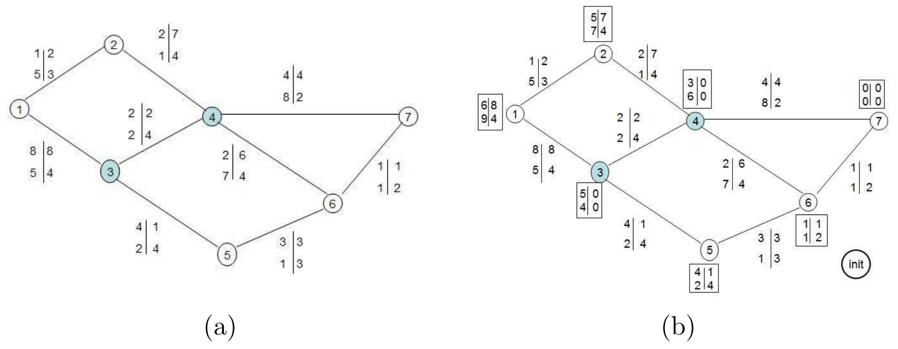


Figure 2-9: (a) An example network, (b) Initialization step

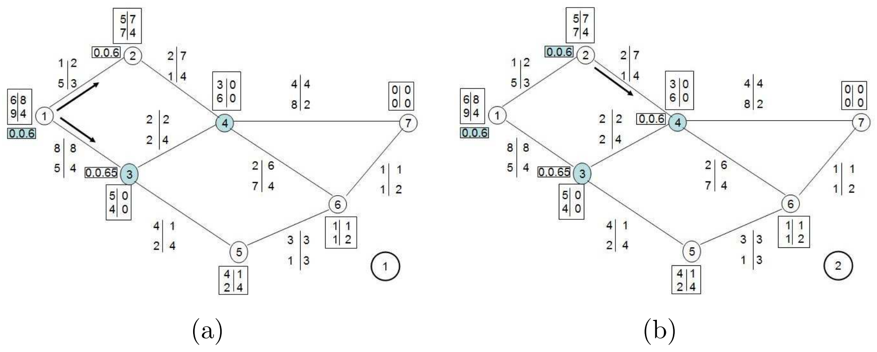


Figure 2-10: (a) Step 1 (b) Step 2

dicted length $l(p^1_{1\to2}) = l(\overrightarrow{q}(p^1_{1\to2}) + \overrightarrow{b}(2))$= $\max(\frac{1+5}{20}, \frac{5+7}{20})$= 0.6, a regenerator count $reg\_numb(p^1_{1\to2}) = 0$, the length of the physical impairment vector $l^*(p^1_{1\to2})$=$\max(\frac{2}{10}, \frac{3}{10})$= 0.3, and shortest impairment distance to the nearest regenerator node (or to t) $look\_ahead\_p(p^1_{1\to2}) = max(\frac{2+7}{10}, \frac{3+4}{10}) = 0.9$. The path to node 3 has $l(p^1_{1\to3})$=0.65, $reg\_numb(p^1_{1\to3}) = 0$, $l^*(p^1_{1\to3})$=0.8, and $look\_ahead\_p(p^1_{1\to3}) = 0.8$. Both of them are stored in the queue because their predicted lengths are less than $maxlen\_q$, the lengths of the physical impairment vectors and the $look\_ahead\_p$ values are less than $maxlen\_p$. The numbers shown in the small boxes are the regenerator count and the predicted QoS length of the (sub)paths stored at the node.

Step 2 (Figure 2-10(b)): We have two subpaths in the queue ($p^1_{1\to2}$ and $p^1_{1\to3}$) with equal regenerator count. But node 2 has the minimum length subpath ($l(p^1_{1\to2} = 0.6)$), and is extracted next. Its neighbors are nodes 1 and 4. But node 1 is the previous node, so it should not be scanned. For node 4, path $p^1_{1\to4} = 1 \to 2 \to 4$, with $l(p^1_{1\to4})$=0.6, $reg\_numb(p^1_{1\to4})$=0, $l^*(p^1_{1\to4})$=0.9, and $look\_ahead\_p(p^1_{1\to4}) = 0.9$ is added to the queue.

Step 3 (Figure 2-11(a)): Both of the subpaths in the queue have zero regenerator count. Then node 4, which has the minimum QoS length subpath ($l(p^1_{1\to4}) = 0.6$), is extracted from the queue. All its neighbors, except node 2, which is the previous node, are

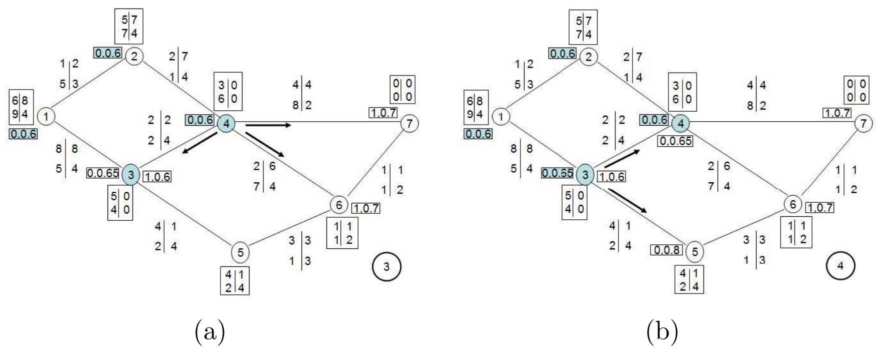


Figure 2-11: (a) Step 3 (b) Step 4

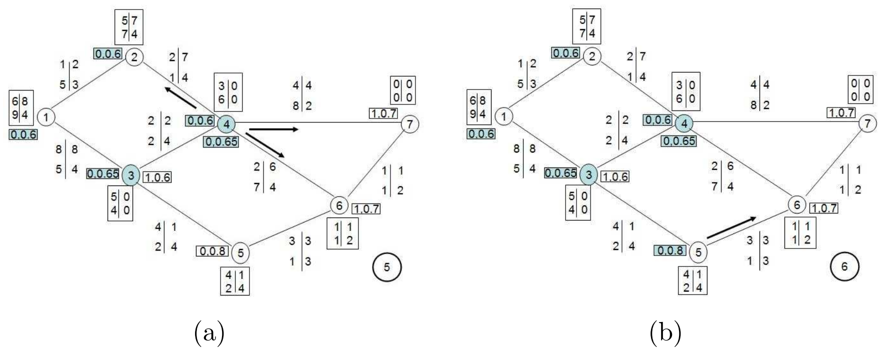


Figure 2-12: (a) Step 5 (b) Step 6

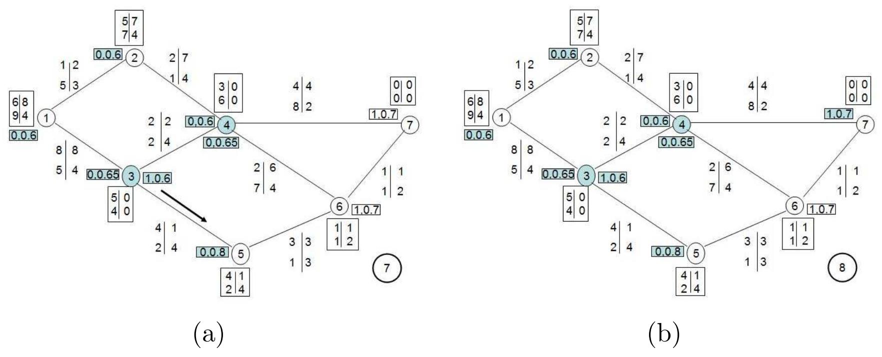


Figure 2-13: (a) Step 7 (b) Step 8

scanned. For node 3, a second path, $p^2_{1\to3} = 1 \to 2 \to 4 \to 3$, with QoS predicted length $l(p^2_{1\to3}) = 0.6$, $reg\_numb(p^2_{1\to3})$=0, $\overrightarrow{I}(p^2_{1\to3}) = (11, 11)$, and $look\_ahead\_p(p^2_{1\to3}) = 0.4$ is found. However, since $l^*(p^2_{1\to3}) = 1.1 > maxlen\_p$, a regeneration at node 4 is required. After regeneration, the history regarding the physical impairment is reset. Thus, $reg\_numb(p^2_{1\to3}) = 1$, and $\overrightarrow{I}(p^2_{1\to3}) = (2, 4)$. Similarly, for nodes 6 and 7 regeneration is required for their respective subpaths at node 4. After regeneration, the path at node 6 is $p^1_{1\to6} = 1 \to 2 \to 4 \to 6$ with $l(p^1_{1\to6}) = 0.7$, $reg\_numb(p^1_{1\to6}) = 1$, $l^*(p^1_{1\to6}) = 0.6$, and $look\_ahead\_p(p^1_{1\to6}) = 0.7$; where as the path at node 7 is $p^1_{1\to7} = 1 \to 2 \to 4 \to 7$, with $l(p^1_{1\to7}) = 0.7, reg\_numb(p^1_{1\to7}) = 1$, $l^*(p^1_{1\to7}) = 0.4$, and $look\_ahead\_p(p^1_{1\to7}) = 0.4$. All of the paths obey the constraints and they are stored in the queue. Since node 7 is the destination node, and since $reg\_count(p^1_{1\to7}) < max\_reg\_count$, we set $max\_reg\_count = reg\_count(p^1_{1\to7}) = 1$.

Step 4 (Figure 2-11(b)): Node 3, which has the minimum regenerator count subpath ($reg\_count(p^1_{1\to3}) = 0$), is extracted. All of its neighbors except node 1, which is the previous node, are scanned. The path to node 4 has a predicted length $l(p^2_{1\to4}) = 0.65$, a regenerator count $reg\_numb(p^2_{1\to4}) = 0$, a physical impairment vector length $l^*(p^2_{1\to4})$=1.0, and $look\_ahead\_p(p^2_{1\to4}) = 0.4$. The path to node 5 has $l(p^1_{1\to5})$=0.8, $reg\_numb(p^1_{1\to5}) = 0$, $l^*(p^1_{1\to5})$=0.9, and $look\_ahead\_p(p^1_{1\to5}) = 0.8$. Since both paths obey the constraints, they are stored in the queue.

Step 5 (Figure 2-12(a)): In this step, there are two subpaths in the queue with the minimum regenerator count ($p^2_{1\to4}$ and $p^1_{1\to5}$). Since node 4 has a subpath with the minimum length ($l(p^2_{1\to4}) = 0.65$), it is extracted. Except for node 3 which is the previous node, all its neighbors are scanned. The path to node 2 $p^2_{1\to2} = 1 \to 3 \to 4 \to 2$ is discarded because $look\_ahead\_p(p^1_{1\to5}) = 1.4 > maxlen\_p$. The new path to node 6 ($p^2_{1\to6}$) has $l^*(p^2_{1\to6}) = 1.6 > maxlen\_p$, hence regeneration is required at node 4. The path is discarded even after regeneration because its regenerator count $reg\_numb(p^2_{1\to6}) = 1 >= max\_reg\_count$. The new path to node 7 ($p^2_{1\to7}$) also is not stored because its regenerator count is equal to max_reg_count.

Step 6 (Figure 2-12(b)): In this step, node 5 is extracted because it has a subpath with the minimum regenerator count ($reg\_numb(p^1_{1\to5}) = 0$). Here we find a second subpath to node 6 with $l(p^2_{1\to6}) = 0.8$, $reg\_numb(p^2_{1\to6}) = 0$ and $l^*(p^2_{1\to6}) = 1.2$. However, since $l^*(p^2_{1\to6}) = 1.2 > maxlen\_p$, a regeneration at node 3 is required. After regeneration, $reg\_numb(p^2_{1\to6}) = 1 >= max\_reg\_count$, and hence is discarded.

Step 7 (Figure 2-13(a)): In this step, we have many subpaths with the minimum regenerator count. Among them, the second path at node 3, $p^2_{1\to3}$ with $l(p^2_{1\to3}) = 0.6$, has the minimum predicted length and is extracted from the queue. Since node 1 is already in the the extracted path, only node 5 is scanned. Yet we do not store the second path at node 5 $p^2_{1\to5}$ because its regenerator count $reg\_numb(p^2_{1\to5}) = 1 >= max\_reg\_count$.

Step 8 (Figure 2-13(b)): We have two subpaths, $p^1_{1\to6}$ and $p^1_{1\to7}$ in the queue with the same regenerator count and predicted QoS length; however, we extract the path stored at node 7 $p^1_{1\to7}$ because it has a smaller physical impairment length $l^*(p^1_{1\to7}) = 0.4$. Since node 7 is the destination node, the extraction process terminates and the path $p^1_{1\to7} = 1 \to 2 \to 4 \to 7$ is returned as a solution.

2-6 Heuristic Algorithms

The impairment-aware QoS routing (IQR) problem is NP-complete. The exact algorithms presented in the previous sections have a factorial time complexity. Thus, it is desirable to find heuristic algorithms which have better time performance than the exact algorithm. In this section, we provide two types of heuristic algorithms to solve each of the two variants of the IQR problem. The first type is obtained by modifying the exact algorithm in such a way that the maximum number of paths stored at a node (k_{max}) is tuned/limited to some value. Limiting k_{max} attains a better time performance at the possible loss of exactness. We refer to the tunable heuristic algorithm obtained by restricting the value of k_{max} for the QoS optimization objective and the regenerator-count minimization objective as TIOQRA and TIQRRM, respectively. The time complexity of these algorithms is polynomial which can be obtained from Equation 2-3 by merely replacing k_{max} by a constant value.

In the rest of this section, we present the second type of our heuristic algorithms which are based on the tabu-search approach. Before we describe the algorithms, we give a brief overview of tabu search in the following section.

2-6-1 Tabu-Search Overview

Tabu search (TS) was introduced by Glover [44] as a general iterative meta-heuristic for solving combinatorial optimization problems. TS is an iterative meta-heuristic which guides simpler heuristics in such a way that they explore various areas of the solution space and prevents them from remaining in local optima. In every iterative step of the TS method, we begin with some current solution and explore its neighboring solutions. A solution's neighborhood, $H(i, k)$, is the set of all solutions that can be transitioned to from the current solution i at iteration k by applying some elementary transformation to the current solution. A move is an operation by which one solution transitions into a neighboring solution. The final result of the TS, called the *incumbent solution*, is obtained by selecting the best solution found overall after executing the search for a desired number of iterations.

TS maintains a list called *tabu list* to prevent the search technique from getting stuck in a local optimum or cycling between already seen solutions. The tabu list records a certain number of previous moves which are then forbidden for as long as they remain in the list. After every iteration, the tabu list is updated by adding the current move to the list and removing the oldest element if the list is full.

If we denote the incumbent solution by i^* and the objective function of the problem by $f()$, the basic elements of TS can be summarized in six steps as follows:

1. Create an initial solution i at random. Set $i^* = i$ and $k = 0$.
2. Set $k = k + 1$ and generate a move (depending on a strategy specific to the problem) to a neighboring solution until a move that is not in the tabu list is obtained.
3. Let j be the solution obtained by the k^{th} move. Set $i = j$.
4. If $f(i) < f(i^*)$ then set $i^* = i$.
5. Update the tabu list.
6. If a stopping condition is met then stop. Else go to 2.

Sometimes, intensification and diversification strategies are used to improve the search. In intensification, the search is emphasized in the promising regions of the feasible domain; whereas in diversification, an attempt is made to consider solutions in a broad area of the search space. Generally, a key to developing a good tabu search algorithm is to define a good initial solution, neighborhood structure, and evaluation function.

2-6-2 The Tabu-search Based Heuristic Algorithm

An interesting insight into the IQR problem is that a local improvement on a partial path leads to improvement of the overall path. If we start with a path P from the source s to the destination t, we can improve the overall length of the path by locally improving the length of a partial path of P. By performing this operation repeatedly for a certain amount of iterations, a good suboptimal solution can be found. This nature of the problem makes the tabu-search approach an efficient method to solve it. In addition, the capability provided by the tabu list can be used to avoid selecting the same partial path repeatedly, thereby enabling us to improve all the possible different partial paths. Furthermore, the diversification strategy of the tabu search method capacitates us to search for the solution in a broader area of the search space. Moreover, the TS approach is shown to be efficient technique to solve the related problems in [48] and [49]. The idea of iteratively improving on a partial path is used in [49] for QoS routing in electronic networks. We have modified this idea to fit the impairment aware QoS routing problem.

Our tabu-search based algorithm was designed to iteratively improve the length of the path P from s to t by improving the length of a partial path F of P. This improvement is achieved by replacing old partial path F with a new partial path $\bar{F}$ of lower length. The outline of the searching process of our heuristic algorithm is as follows:

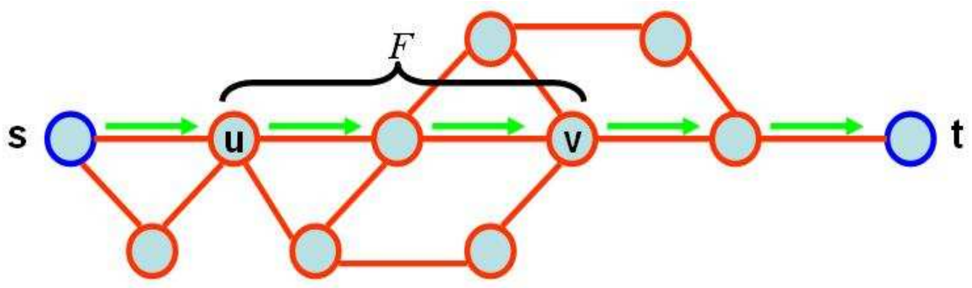
1. We initialize P to the shortest path from s to t, $p_{s\to t}$ computed using Dijkstra's Algorithm by imposing one QoS metric.

2. A pair of nodes u and v is selected on path P such that $P = p_{s\to t} = p_{s\to u}+p_{u\to v}+p_{v\to t}$ as shown in Figure 2-14(a). The hop count of the selected partial path $p_{u\to v}$ should not exceed a given small integer L, because the procedure outlined in step (3) can rebuild a new path from u to v in a short time for a small L.

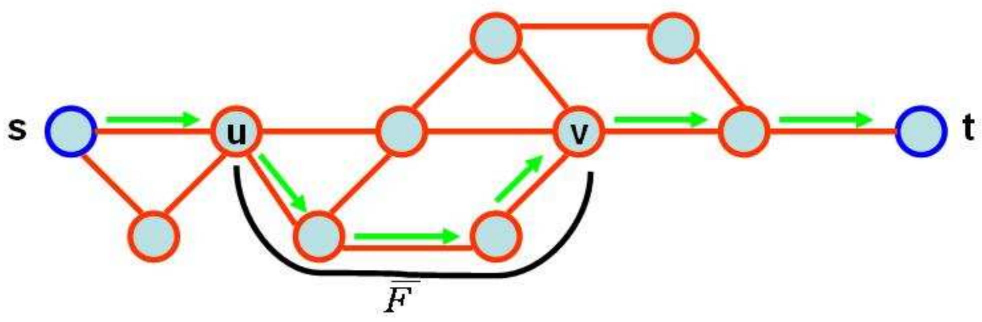
3. A new partial path $\bar{F}$ from u to v is searched by the subroutine IMPROVE_Q (Algorithm 5) or the IMPROVE_R (Algorithm 7) depending on our objective of minimization. These procedures build a state-space tree which facilitates the searching process to find a new $\bar{F}$ with a smaller length. If no path $\bar{F}$ with a lower length than F is found, the procedures return an empty value.

4. Let $\bar{P} = p_{s\to u} + \bar{F} + p_{v\to t}$ as shown in Figure 2-14(b). Set $P = \bar{P}$ and jump back to step (1) if the stop condition is not met.

5. Return the best path found after performing the iterations.



(a) Selecting a partial path.



(b) Replacing the old partial path with a better one.

Figure 2-14: Example of partial path replacement.

A path rebuilding procedure can be built by putting together steps (1) to (4). This procedure can be embedded into a tabu-search based iteration loop in which a tabu list is implemented by a circular queue for storing all the selected partial paths $p_{u\to v}$. Only partial paths that are not in the tabu list are eligible to be selected for path rebuilding. The tabu-search based procedure stops when the desired number of iterations have elapsed or when the tabu-search based path-rebuilding procedure does not improve the length of P after some fixed consecutive iterations.

In our tabu-based heuristic algorithm, we maintain the non-linear definition of path length described in Section 2-5. In addition, some of the parammeters listed in Section 2-2 are used. In the following sections, the detailed description of our tabu-search based heuristic algorithms for the two objectives of minimization are presented.

2-6-3 QoS Optimization

We refer to our tabu based heuristic algorithm to solve the IOQR problem as TABU_Q. TABU_Q starts by computing an initial path P which is the shortest path from the source node s to the destination node t in terms of one QoS metric. It then tests if P is feasible path or not. P is said to be feasible if it satisfies both the QoS and the physical constraints. If P is feasible, then a random partial path F is chosen for improvement. But if P is not feasible, a partial path is selected in the vicinity of the node where either of the constraints is violated for the first time.

Once a partial path is selected, the IMPROVE_Q subroutine is called to find a better partial path $\bar{F}$ that can replace F. If P is feasible, the IMPROVE_Q subroutine tries to find a partial path that minimizes the QoS length. But if P is not feasible, IMPROVE_Q tries to find a partial path that satisfies the violated constraint. IMPROVE_Q does so by constracting a state space tree. If IMPROVE_Q is successful in finding a better partial path, then P is updated by replacing the old partial path F by the new partial path $\bar{F}$. Then the feasiblity of P is tested and the process of selecting and replacing a partial path continues. The iteration stops when the total number of iterations is elapsed, or if the path P is not improved after a certain number of consecutive number of iterations. Finally, TABU_Q returns the feasible path (if any is found) with the smallest QoS length, otherwise it returns an empty path. The feasible path with the best length is denoted by the parammeter *Best_path*.

To avoid the searching of paths too far away from the optimal solutions, a restart strategy is implemented in the following way: an original path *ori_path* replaces the current path P for the next iteration. This is triggered if an empty path $\bar{F}$ is returned by IMPROVE_Q for a number of consecutive iterations. The original path *orig_path* saves the path before the last improvement.

Based on the above ideas, the complete pseudo-code of TABU_Q is developed in Algorithm 4. TABU_Q starts by initializing P to the shortest path from s to t computed using Dijkstra's algorithm by imposing the first QoS metric (line 2). *Best_path* represents

Algorithm 4 TABU_Q(G, m_p, m_q, s, t)

```
1: Let $s$ denote the source node, and $t$ denote the destination node;
2: Let $P$ be a path from $s$ to $t$, and $P$ is determined by Dijkstra's algorithm based on the first QoS metric;
3: if P is a feasible path then {* $Best\_path$ is the feasible path with the best length so far *}
4:     $Best\_path = P$; $feasible = 1$;
5: else
6:     $Best\_path = \emptyset$; $l(Best\_path) = \infty$; $feasible = 0$;
7: end if
8: Initialize a circular queue $Z$ to be the tabu list;
9: Given two small integers $L_{min}$ and $L_{max}$, $L = L_{min}; j = r = h = y = 0$;
10: while $(k < ITERATIONS)$ do
11:     if $feasible = 0$ then
12:         Let node $x$ be the earliest node where path $P$ fails either of the constraints;
13:         Set $flag = 0$ if $P$ violates the QoS constraint, and $flag = 1$ if $P$ violates the physical impairment at node $x$;
14:         Select a partial path $F$ of $P$ in the vicinity of node $x$, where $F \subset P$ and $|F| \leq L$;
15:     else
16:         $flag = 0$;
17:         Select a random partial path $F$ of $P$, where $R \subset P$ and $|F| \leq L$;
18:     end if
19:     Assume $P = P_1 + F + P_2$ and $F$ is a path from node $u$ to node $v$, $P_1 = p_{s \to u}$ and $P_2 = p_{v \to t}$;
20:     if $(F \notin Z)$ then {* if $F$ is not in the tabu list *}
21:         $Z = Z \cup F$;    {* insert $F$ into the tabu list *}
22:         Let $D$ be an $1 \times N$ array whose entry at $i^{th}$ index is 1 if node $i$ has appeared in either $P_1$ or $P_2$; 0 otherwise;
23:         $\bar{F} = IMPROVE\_Q(G, s, u, v, l(F), l(P), D, flag)$; {* compute $\bar{F}$ *}
24:         if $(\bar{F} \neq \emptyset)$ then {* if a better partial path is found *}
25:             $L = L_{min}$; $orig\_path = P$;    {* $orig\_path$ is used to avoid searching too far away from the optimal
                solution *}
26:             Rebuild a new path $P$ from node $s$ to node $t$ such that $P = P_1 + \bar{F} + P_2$;
27:             if $P$ is a feasible path then
28:                 $feasible = 1$;
29:                 if $l(P) < l(Best\_path)$ then {* if a better feasible path found *}
30:                     $Best\_path = P$; $y = 0$;    {* update $Best\_path$ *}
31:                 else
32:                     $y++$;
33:                 end if
34:             else
35:                 $feasible = 0$; $y++$;
36:             end if
37:             $h = 0$;
38:         else
39:             $y++$; $r++$;
40:             if $((r > ITERATIONS/10)$ and $(L \leq L_{max}))$ then {* if $\bar{F}$ is not found for $r$ consecutive iterations
                *}
41:                 $L++$; $r = 0$;    {* increment the hop length of the selected partial path $F$ *}
42:             end if
43:             $h++$;
44:             if $h > ITERATIONS/5$ then {* if $P$ is not improved for $h > ITERATIONS/5$ consecutive
                iterations *}
45:                 $P = orig\_path$; $h = 0$;    {* restart the search from $orig\_path$ *}
46:             end if
47:         end if
48:     end if
49:     if $y > ITERATIONS/2$ then {* if $P$ is not improved for $y > ITERATIONS/2$ consecutive iterations
        *}
50:         RETURN $Best\_path$;    {* stop the iteration and return the best path found so far *}
51:     end if
52: end while
53: RETURN $Best\_path$;
```

the path with the best QoS length among all the feasible paths that are found so far. If P is feasible, then *Best_path* is initialized to P in line 4. The variable $feasible$ remembers whether the current path P is feasible or not.

Once we have initialized P, the next task is to select a partial path F of P. We perform the subpath selection process in two different ways depending on whether P is feasible or not. If P is feasible, we assume that all subpaths of P are equally important; hence, F is selected randomly (line 17). If P is feasible, the IMPROVE_Q function targets at finding a path $\bar{F}$ that minimizes the length in terms of the QoS metrics. But if P is not feasible, then F is selected in the vicinity of the node x where path P for the first time violates either of the QoS constraints or the physical constraints (line 14). This helps us to target our improvement on the local area of P where the constraints are violated. If the QoS constraint is violated at node x, then IMPROVE_Q targets at minimizing the length in terms of the QoS metrics; but if the physical impairment is violated, IMPROVE_Q targets at minimizing the length in terms of the physical impairment since the last regenerator ($\overrightarrow{I^*}$). The variable $flag$ (lines 13,16) is used to 'inform' IMPROVE_Q the current target of optimization. This differentiated treatment offered by IMPROVE_Q increases the efficiency of the algorithm.

Next, if the selected partial path F is not in the tabu list, then it is added to the tabu list (line 21). Let u and v be the source node and the destination node of F, respectively. Let P_1 is the partial path from s to u ($p_{s\to u}$), and P_2 is the partial path from v to t ($p_{v\to t}$). In order to avoid loops, we remember (in D) the list of nodes that are already used in the partial paths P_1 and P_2, so that none of them will be used again in computing $\bar{F}$ (line 22). The partial path F is passed to the IMPROVE_Q function along with the necessary parameters as listed in line 23. When IMPROVE_Q fails to find a path with a shorter length, it returns an empty path. If IMPROVE_Q returns a path $\bar{F} \neq \emptyset$, a new path P will be rebuilt by replacing F with $\bar{F}$ (line 26). If the new path P is feasible and has a smaller length than *Best_path*, then *Best_path* is updated to P (line 30).

The number of hops L of the selected partial path F changes dynamically during the iterations to allow flexibility; thereby attaining better performance. At line 9, the initial value of L is set to be L_{min}, and the value is increased by 1 at line 41 if IMPROVE_Q returns empty path for a certain number of consecutive iterations. The value of L is reset back to L_{min} at line 25 if a path $\bar{F} \neq \emptyset$ is returned by IMPROVE_Q. Note that the maximum value of L is set to a small integer, L_{max}, because the function IMPROVE_Q is efficient only for small value of L. Lines 25 and 45 are used to handle the restart strategy to avoid searching far away from the optimal solution. Furthermore, if the length of the *Best_path* is not improved after a fixed amount of iterations, we return the current *Best_path* as a solution in line 50.

The pseudo-code of the IMPROVE_Q subroutine is presented in Algorithm 5. The

Algorithm 5 IMPROVE_Q$(G, s, u, v, l_F, l_P, D, flag)$

```
1: Let $Best\_len = l_F$;   {* $Best\_len$ stores the best length obtained so far and is initialized to the QoS length of $F$ *}
2: Let $Best\_slr = l^*(\overrightarrow{I^*}(p_{s\to v}))$;   {* $Best\_slr$ stores the best length of the physical impairments since the last regenerator, and is initialized to the corresponding value at the destination node of $F$ *}
3: $\bar{F} = \emptyset$, $node\_count = 0$;   {* $node\_count$ represents the number of nodes inserted into the tree *}
4: Let state $S_0$ store the source node $u$ and be the root of the state-space tree, and let $tree\_size$ be the maximum number of nodes that the state space tree can have.
5: Add $S_0$ to an empty node heap $H$;
6: while ($H \neq \emptyset$ AND $node\_count \leq tree\_size$) do
7:   $S_A = EXTRACT\_MIN(H)$;   {* extract the path with the minimum measure *}
8:   for each node $B$ adjacent to node $A$ stored at state $S_A$ do
9:     if ($B$ is not on the path from $S_0$ to $S_A$ and $B$ is unvisited from $A$ and $D[B] = 0$) then
10:      create a new state node $S_B$ of $A$ based on the information stored in state node $S_A$ of $A$;
11:      $S_B \to path = S_A \to path + (A, B)$;   {* extend the path to $B$ *}
12:      Let $P_B$ denote the path from $S_0$ to $S_B$;
13:      if (($flag = 1$ AND $l(P_B) - l_F > maxlen\_q - l_P$) OR ($flag = 0$ AND $l(P_B) > Best\_len$) OR if the path up to node $B$ violates the physical impairment even after regeneration(if possible)) then {* test if the extended path is valid to be stored *}
14:        discard $B$ and continue with the next neighbor of $A$.
15:      else
16:        if $B = v$ then {* if $B$ is the destination node of $F$ *}
17:          if $flag = 0$ then {* if the current target of optimization is the QoS length *}
18:            if $l(P_B) < Best\_len$ then
19:              $Best\_len = l(P_B)$;   {* update the best length to the length of $P_B$ *}
20:              $\bar{F} = P_B$;   {* update $\bar{F}$ to $P_B$ *}
21:            end if
22:          else {* if the current target of optimization is the physical impairment length *}
23:            if $l^*(P_B) < Best\_slr$ then
24:              $Best\_slr = l^*(P_B)$;
25:              $\bar{F} = P_B$;
26:            end if
27:          end if
28:        end if
29:        if $B \neq v$ then
30:          Add $B$ to heap $H$;
31:        end if
32:        Increment $node\_count$ and make the new state node $S_B$ be a child of state node $S_A$;
33:      end if
34:    end if
35:  end for
36: end while
37: return $\bar{F}$;
```

purpose of the subroutine is to find a better path $\bar{F}$ that replaces the partial path F. The subroutine takes the following parameters as arguments: the graph G representing the network topology, the source node of P (s), the source node of F (u), the destination node of F (v), the QoS length of F (l_F), the Qos length of P (l_P), the array D that stores the already used nodes, and the $flag$ that tells which constraint is violated in P. IMPROVE_Q operates by maintaining a state space tree of a fixed size and a heap structure. After initializing the root of the state space tree to u and inserting u into the heap, IMPROVE_Q iteratively extracts a path with the minimum measure from the heap and extends it to the neighboring nodes of the last node in the extracted path. If the extended path is eligible to be stored, then it is stored in the heap. Then the new neighbor node is inserted into the state space tree as a child of the last node in the extracted path. The insertion into and extraction from the heap continues as long as the heap is not empty and the maximum size of the state space tree is not reached. Finally, the path from u to v with the best measure is selected. If this path improves F, then it is returned as $\bar{F}$. Otherwise an empty path is returned.

First, the *Best_len*, which stores the QoS length of the best path encountered so far, is initialized to the QoS length of F, l_F (line 1). The *Best_slr* stores the length of the physical impairment since the last regenerator ($\overrightarrow{I^*}$) of the best partial path, and is initialized to the corresponding value at the destination node v of F (line 2). As mentioned before, we try to find a path $\bar{F}$ attaining a smaller *Best_len* if $flag = 0$, and a smaller *Best_slr* if $flag = 1$. In line 3, $\bar{F}$ is initialized to an empty path. In addition, the variable *node_count*, that registers the number of nodes that have been added to the state space tree, is initialized to 0. The maximum number of nodes that can be inserted into the state space tree is limited by *tree_size*.

We start by creating a state S_0 which is the root of the state space tree. After storing the source node u of F, S_0 is added to the heap H (lines 4-5). The state-space tree is then constructed by the while loop by adding one new node at a time. As long as the heap H is not empty and the tree is not full, the state S_A storing node A with the minimum measure is extracted (lines 6 and 7). If $flag = 0$, then the IMPROVE_Q function extracts the one with the smallest QoS length; otherwise, it extracts the one having the smallest length with respect to the physical impairment since the last regenerator node ($\overrightarrow{I^*}$). Next, each neighbor B of node A that is not in D and also is not involved in the subpath from u to A is scanned (line 9). This check is performed in order to avoid loops. The neighbor node B, along with its new extended path from node A, is stored in a new state S_B (lines 10-12).

In order for S_B to be inserted into the tree, the new extended path P_B has to pass the following tests (line 13). Firstly, the cumulative path $P_1 + P_B$ should satisfy the physical constraint. Secondly, since it does not make sense to store a path with a larger length than the best length known so far, the length $l(P_B)$ should not exceed *Best_len* if $flag = 0$. If $flag = 1$, we should make sure that the difference between the QoS lengths of P_B and F ($l(P_B) - l_F$) does not exceed the difference between $maxlen\_q$ and the QoS length of P ($maxlen\_q - l_P$); otherwise P_B will lead to a path that will violate the QoS constraint. If any of the conditions in line 13 are true for the new extended path P_B, then the current node B is not eligible to be added to both the tree and the heap. But if P_B passes the test in line 13, then S_B is added to the tree as a child of S_A incrementing $node\_count$, and B is added to the heap if $B \neq v$ (lines 29-32).

If B is the destination node v, then either $Best\_len$ or $Best\_slr$ are updated depending on the value of $flag$, and $\bar{F}$ is set to P_B if P_B is better than $\bar{F}$ (lines 16-28). Note that, $\bar{F}$ stores the best path obtained thus far. Finally, when the heap H is empty, $\bar{F}$ is returned as a best solution in line 37.

Complexity Analysis

In this section we present the time complexity of our algorithm. First, we present the complexity of the IMPROVE_Q subroutine listed in Algorithm 5. The insertion and extraction from the heap H can be performed in O(1), whereas, the check performed in line 9 on node B to avoid loop takes O(N), where N is the number of nodes in the network. In addition, the path computations in lines 11 and 12 are also O(N) at the worst. Further, the test conditions in line 13 take O($N(m_q + m_p)$), where m_q and m_p represent the number of the QoS metrics and the physical impairments, respectively. Lines 18 and 19 are O(1) because $l(P_B)$ is already calculated in line 13. Computing the length of the vectors in line 23 takes O(m_p). Copying the path P_B on F in lines 20 and 25 also takes O(N). All the other operations are O(1). If d is the maximum degree in the network, the for loop in line 8-35 is invoked at most d times for each extracted node. Moreover, the while loop that starts in line 6 is invoked at most $tree\_size$ times, where $tree\_size$ is the maximum number of nodes that can be inserted into the state space tree. Therefore, the overall complexity of the IMPROVE_Q function is O($tree\_size \times d \times (1 + N + N(m_q + m_p))$) = O($tree\_size \times d \times N(m_q + m_p)$).

The complexity of the main algorithm listed in Algorithm 4 is given as follows. Executing the Dijkstra function in line 2 takes O($|E|logN$) times, where $|E|$ is the number of edges in the network. The feasibility checks in lines 3 and 27 lead to O($N(m_q + m_p)$). The assignments in line 4 are O(N). Finding the node x in line 12 leads to O($N(m_q+m_p)$) time. The selection of the subpath F in lines 14 and 17 are both O(L_{max}), where L_{max} is the maximum possible hop length of F. Checking whether F is in the queue Z or not in line 20 takes O($size\_Z \times L_{max}$) time, where $size\_Z$ is the size of the queue. It takes O(L_{max}) to add F into Z in line 21. Line 22 is O(N). Executing the IMPROVE_Q function in line 23 adds O($tree\_size \times d \times N(m_q+m_p)$). Lines 25, 26, 30 and 45 are O(N). The others are are O(1).

Therefore, the overall complexity of the algorithm is: O($1 + |E|logN + N(m_q + m_p) + N + k(N(m_q + m_p) + N + 1 + L_{max} + size\_Z \times L_{max} + tree\_size \times d \times N(m_q + m_p)$),

where k is the number of iterations.
With $m_p = m_q = m$, the overall complexity is given as

$$C_{TABU\_Q} = O(|E|logN + k(L_{max})(size\_Z) + kmdN(tree\_size)) \tag{2-4}$$

Since k, L_{max}, $size\_Z$, and $tree\_size$ are constants, the time complexity of TABU_Q is polynomial.

2-6-4 Regenerator-Count Minimization

In this section, we present a tabu based heuristic algorithm to solve the IQRRM problem defined in Section 2-5-2. We refer to our tabu-based heuristic algorithm as TABU_R, and it is listed in Algorithm 6. TABU_R is a modification of TABU_Q described in Section 2-5-2. Like TABU_Q, TABU_R also starts with an initial path P that is the shortest path from the source node s to the destination node t in terms of one QoS metric. It then selects a partial path F of P in the same way as TABU_Q depending on the feasibility of P. After selecting a partial path F, the IMPROVE_R subroutine searches for a better partial path $\bar{F}$ that replaces F. If P is feasible, the IMPROVE_R subroutine tries to find a partial path that minimizes the regenerator-count. Otherwise, IMPROVE_R tries to find a partial path that satisfies the violated constraint in the same way as the IMPROVE_Q subroutine. If IMPROVE_R finds a better partial path, then P is updated by substituting $\bar{F}$ in place of F. The iteration then continues by selecting and replacing a partial path at a time. After performing the iterations, TABU_R returns the feasible path (if any is found) with the least regenerator-count. If no feasible path is found, then it returns an empty path. The feasible path with the best regenerator-count is represented by $Best\_path$.

The operations involved in TABU_Q and TABU_R are similar in various ways because of the similarity of the problems they solve. Thus, most of the codes in the pseudo-code of TABU_R are inherited from TABU_Q. In this section, we explain only the codes that are peculiar to TABU_R to avoid repetition. The two algorithms differ in the criteria they use to choose the best path, and the functions they use to compute the partial path $\bar{F}$ that replaces F. In line 4 of TABU_R, $Best\_regcount$, which keeps track of the regenerator count of the feasible path with the smallest regenerator count, is initialized to the regenerator count of the initial path P. If a new feasible path with a smaller regenerator count is found, $Best\_path$ and $Best\_regcount$ are updated to the new path and its regenerator count, respectively (lines 29 and 30).

The subroutine IMPROVE_R called in line 23 of TABU_R also inherits most of its codes from IMPROVE_Q, and we explain only the codes that are special to IMPROVE_R to avoid repetition. The meta-code of IMPROVE_R is given in Algorithm 7. In line 3, the $Best\_regcount$ is initialized to the number of regenerators used in the parent path P up to the destination node v of F. The other operations in lines 1-15 are taken from IMPROVE_Q. If the new neighbor node B of A is the destination node v (line 18), $\bar{F}$

Algorithm 6 TABU_R(G, m_p, m_q, s, t)

```
1: Let $s$ denote the source node, and $t$ denote the destination node;
2: Let $P$ be a path from $s$ to $t$, and $P$ is determined by Dijkstra's algorithm based on the first QoS metric;
3: if P is a feasible path then {* $Best\_path$ is the feasible path with the best regenerator-count so far *}
4:     $Best\_path = P$; $Best\_regcount = reg\_numb(P)$; $feasible = 1$;
5: else
6:     $Best\_path = \emptyset$; $Best\_regcount = \infty$; $feasible = 0$;
7: end if
8: Initialize a circular queue $Z$ to be the tabu list;
9: Given two small integers $L_{min}$ and $L_{max}$, $L = L_{min}; j = r = h = y = 0$;;
10: while $(k < ITERATIONS)$ do
11:     if $feasible = 0$ then
12:         Let node $x$ be the earliest node where path $P$ fails either of the constraints;
13:         Set $flag = 0$ if $P$ violates the QoS constraint, and $flag = 1$ if $P$ violates the physical impairment at node $x$;
14:         Select a partial path $F$ of $P$ in the vicinity of node $x$, where $F \subset P$ and $|F| \leq L$;
15:     else
16:         $flag = 0$;
17:         Select a random partial path $F$ of $P$, where $R \subset P$ and $|F| \leq L$;
18:     end if
19:     Assume $P = P_1 + F + P_2$; $F$ is a path from node $u$ to node $v$, $P_1 = p_{s \to u}$, and $P_2 = p_{v \to t}$;
20:     if $(F \notin Z)$ then {* if $F$ is not in the tabu list *}
21:         $Z = Z \cup R$;    {* insert $F$ into the tabu list *}
22:         Let $D$ be an $1 \times N$ array whose entry at $i^{th}$ index is 1 if node $i$ has appeared in either $P_1$ or $P_2$; 0 otherwise; and
           let $v_{rc}$ be the number of regenerators used along $p_{s \to v}$.
23:         $\bar{F} = IMPROVE\_R(G, s, u, v, l(F), l(P), D, flag, v_{rc})$; {* compute $\bar{F}$ *}
24:         if $(\bar{F} \neq \emptyset)$ then {* if a better partial path is found *}
25:             $L = L_{min}$; $orig\_path = P$;    {* $orig\_path$ is used to avoid searching too far away from the optimal
               solution *}
26:             Rebuild a new path $P$ from node $s$ to node $t$ such that $P = P_1 + \bar{F} + P_2$;
27:             if $P$ is a feasible path then
28:                 $feasible = 1$;
29:                 if $reg\_numb(P) < Best\_regcount$ then {* update $Best\_path$ and $Best\_regcount$ *}
30:                     $Best\_regcount = reg\_numb(P)$; $Best\_path = P$; $y = 0$;
31:                 else
32:                     $y++$;
33:                 end if
34:             else
35:                 $feasible = 0$; $y++$;
36:             end if
37:             $h = 0$;
38:         else
39:             $y++$; $r++$;
40:             if $((r > ITERATIONS/10)$ and $(L \leq L_{max}))$ then {* if $\bar{F}$ is not found for $r$ consecutive iterations
               *}
41:                 $L++$; $r = 0$;    {* increment the hop length of the selected partial path $F$ *}
42:             end if
43:             $h++$;
44:             if $h > ITERATIONS/5$ then {* if $P$ is not improved for $h > ITERATIONS/5$ consecutive
               iterations *}
45:                 $P = orig\_path$; $h = 0$;    {* restart the search from $orig\_path$ *}
46:             end if
47:         end if
48:     end if
49:     if $y > ITERATIONS/2$ then {* if $P$ is not improved for $y > ITERATIONS/2$ consecutive iterations
       *}
50:         RETURN $Best\_path$;    {* stop the iteration and return the best path found so far *}
51:     end if
52: end while
53: RETURN $Best\_path$;
```

Algorithm 7 IMPROVE_R$(G, u, v, l_F, v_{slr}, l_P, D, flag, v_rc)$

```
1: Let $Best\_len = l_F$;   {* $Best\_len$ stores the best length from $u$ to $v$ and is initialized to the QoS length of $F$ *}
2: Let $Best\_slr = l^*(\overrightarrow{I^*}(p_{s \to v}))$;   {* $Best\_slr$ stores the best length of the physical impairments since the last regenerator at node $v$, and is initialized to the corresponding value at the destination node of $F$ *}
3: $Best\_regcount = v\_rc$   {* $Best\_regcount$ stores the best regenerator count until node $v$, and is initialized to the corresponding value at the destination node of $F$ *}
4: $\bar{F} = \emptyset$, node_count=0;   {* $node\_count$ represents the number of nodes inserted into the tree *}
5: Let state $S_0$ store the node $u$ and be the root of the state-space tree, and let $tree\_size$ be the maximum number of nodes that the state space tree can have.
6: Add $S_0$ to an empty node heap $H$;
7: while ($H \neq \emptyset$ AND $node\_count \leq tree\_size$) do
8:    $S_A = EXTRACT\_MIN(H)$;   {* extract the path with the minimum measure *}
9:    for each node $B$ adjacent to node $A$ stored at state $S_A$ do
10:      if ($B$ is not on the path from $S_0$ to $S_A$ and $B$ is unvisited from $A$ and $D[B] = 0$) then
11:         create a new state node $S_B$ of $A$ based on the information stored in state node $S_A$ of $A$;
12:         $S_B \to path = S_A \to path + (A, B)$;   {* extend the path to $B$ *}
13:         Let $P_B$ denote the path from $S_0$ to $S_B$;
14:         if (($flag = 1$ AND $l(P_B) - l_F > maxlen\_q - l_P$) OR ($flag = 0$ AND $l(P_B) > Best\_len$) OR if the path $P_1 + P_B$ violates the physical impairment even after regeneration(if possible)) then {* test if the extended path is valid to be stored *}
15:            discard $B$ and continue with the next neighbor of $A$.
16:         else
17:            Let $B_{rc}$ be the number of regenerators used until node $B$.
18:            if $B = v$ then {* if $B$ is the destination node of $F$ *}
19:               if $flag = 0$ AND $feasible = 0$ then {* if the current target of optimization is the QoS length *}
20:                  if $l(P_B) < Best\_len$ then
21:                     $Best\_len = l(P_B)$;   {* update the best length to the length of $P_B$ *}
22:                     $\bar{F} = P_B$;   {* update $\bar{F}$ to $P_B$ *}
23:                  end if
24:               else {* if the current target of optimization is the physical impairment length *}
25:                  if $flag = 1$ AND $feasible = 0$ then
26:                     if $l^*(P_B) < Best\_slr$ then
27:                        $Best\_slr = l^*(P_B)$;
28:                        $\bar{F} = P_B$;
29:                     end if
30:                  end if
31:               else
32:                  if $feasible = 1$ then {* if the current target of optimization is the regenerator-count *}
33:                     if $B_{rc} < Best\_regcount$ then
34:                        $Best\_regcount = B_{rc}$;
35:                        $\bar{F} = P_B$;
36:                     end if
37:                  end if
38:               end if
39:            end if
40:            if $v \neq B$ then
41:               Add $B$ to heap $H$;
42:            end if
43:            Increment $node\_count$ and make the new state node $S_B$ be a child of state node $S_A$;
44:         end if
45:      end if
46:   end for
47: end while
48: return $\bar{F}$;
```

is updated based on the conditions in lines 19-33. If the parent path P is not feasible and violates the QoS constraint, and if the QoS length of P_B is less than the best length obtained so far, $\bar{F}$ and $Best\_len$ are updated to P_B and its QoS length, respectively (lines 19-23). In lines 25 to 30, P_B and $Best\_slr$ are updated if P violates the physical impairment and if P_B improves the length of the best physical impairment since the last regenerator. But if the parent path P is feasible and if the number of regenerators used until node B $B_{rc} < Best\_regcount$, $\bar{F}$ and $Best\_regcount$ are updated as in lines 34-35. Therefore, the computation of $\bar{F}$ depends both on feasibility of P and which of the two constraints are violated in P.

The complexity of TABU_R is the same as that of TABU_Q which is given in Equation 2-4.

Chapter 3

Simulation

In this chapter, we present and analyze the simulation results of the impairment-aware QoS routing algorithms discussed in Chapter 2.

3-1 Simulation Environment

In our simulations, our objective is to find a path for a single connection request. A connection request is represented by a request-id, a source, a destination, QoS constraints, and physical constraints. Each link in a network is associated with one wavelength, two QoS link weights, and two physical impairment link weights which are assigned random values between 0 and 1 by a uniformly distributed random function. The regenerators are also randomly distributed in the network.

In all the simulations, 10,000 requests are generated with randomly selected source and destination nodes. For each request, a new network is generated, thus 10,000 different networks are generated overall. The performance metrics used in our simulations are the *acceptance ratio, running time, path length* in terms of the QoS metrics, and the *regenerator-count* of the path. Acceptance ratio refers to the ratio of the number of accepted requests to the total number of requests. Running time represents the time taken by the algorithm to process the connection request. The regenerator-count of a path is the number of regenerators used to set up the path.

When comparing the algorithms with respect to each of the performance metrics, each of the algorithms should operate on the same network for each request. For each request, a new network is generated and each of the algorithms processes the the request on the network. After repeating this for 10,000 connection requests, the average performance metrics are taken for each algorithm. Therefore, a data point is the average of 10,000 requests. In addition to the average values, the standard deviation is also computed to study the variability of the data. Given a set of data X of size c and mean value of μ, the standard deviation σ is calculated as:

$$\sigma = \sqrt{\frac{1}{c}\sum_{i=1}^{c}(X_i - \mu)^2} \tag{3-1}$$

The same set of connection requests and networks are used for each data point. This is achieved by feeding the same seed (for each data point) to the random functions that generate the requests and the networks.

The simulations are performed on the random networks, the square lattice network, and the ARPANET. A random network with N nodes is specified by a parameter ρ which represents the probability of existence of a link between any pair of nodes in the network. A square lattice network is a network with N nodes where the nodes are connected in a square lattice topology of dimension $n \times n$, where $n = \sqrt{N}$. The ARPANET is a realistic optical network with 28 nodes and 45 bidirectional links. The lattice network and the ARPANET topologies are given in Appendix A.

To understand how the algorithms behave under different situations, the algorithms are simulated under four different scenarios. These are the scenarios of totally loose physical constraints, totally loose QoS constraints, tight QoS constraints and medium QoS constraints. In the next sections, we present and analyze the simulation results obtained under these scenarios.

3-2 Simulation under Totally Loose Physical Constraints

In this simulation scenario, the physical constraints are loosened by assigning a value that no path can possibly violate. For a network with N nodes, a path from a source to a destination in the worst-case traverses all the nodes resulting in $N-1$ links. The maximum value of the i^{th} physical impairment on each link is 1. Thus, we set the physical constraints $\Delta_1 = \Delta_2 = \Delta' = N-1$ so that no (sub)path will violate any of them. As a result, only the effect of the QoS constraints are remaining in the network. The QoS constraints are gradually varied from a very small value (tight constraint) to large value (loose constraint), and the corresponding aforementioned performance metrics are measured. In all the experiments the QoS constraints have the same value, i.e, $T_1 = T_2 = T'$.

The experiment is conducted with the QoS optimization algorithms (EIOQRA, TIOQRA, and TABU_Q) on a lattice network of size $N = 49$. The number of regenerators $N_R = 0$

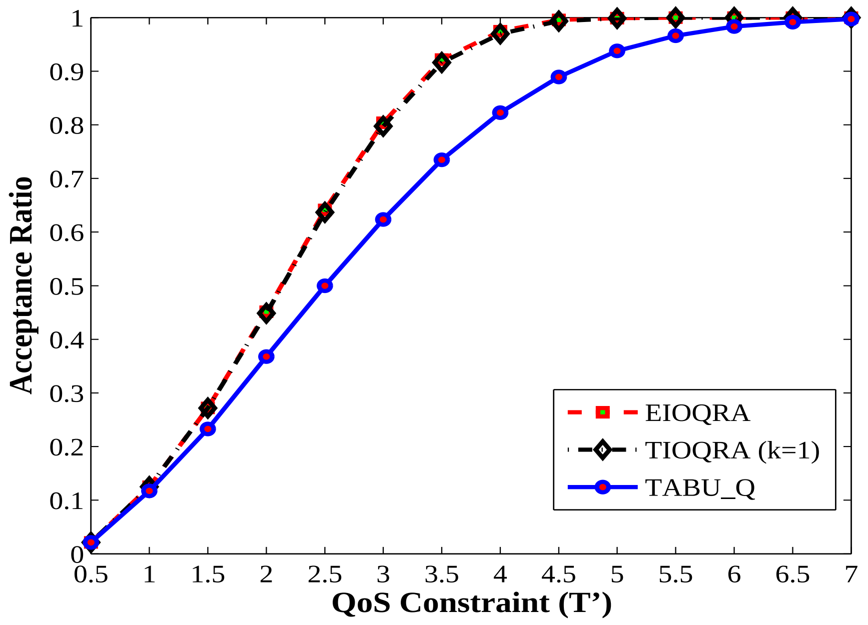


Figure 3-1: Acceptance ratio vs QoS constraint for the lattice network with $N = 49$ under totally loose physical constraints.

because no need of regeneration arises as the physical constraints are totally loose. The maximum number of paths that can be stored at a node is fixed to $k = 1$ for TIOQRA. Figure 3-1 shows the plot of the acceptance ratio of EIOQRA, TABU_Q, and TIOQRA vs the QoS constraint T'. For all the algorithms, the acceptance ratio increases with increasing T'. As larger QoS constraint is used, more paths will be satisfying the constraint, and hence, larger acceptance ratio is obtained. The QoS path length plots of the algorithms are shown in Figure 3-4. As can be seen from the figure, the average path length increases with increase in the Qos constraint. This is because only paths of small QoS length are accepted for small value of T'; but paths with larger QoS lengths are also accepted when T' is larger. At each data point, the standard deviation is computed and it is found to be in the order of the average value. Since the source and the destination nodes of the requests are randomly generated, their lengths are also randomly distributed up to the maximum value. This leads to large standard deviation values at each data point.

In Figure 3-2 is shown the running time plots of the algorithms. The fluctuation in the running time plots is exaggerated because the scale is very small. This very small fluctuation results from the difference in the state of the processor when the different data points are taken. The maximum running time obtained for the algorithms is plotted in Figure 3-3. The same maximum value ($10ms$) is obtained for each algorithm at each value of T'. This is because the running time obtained from the

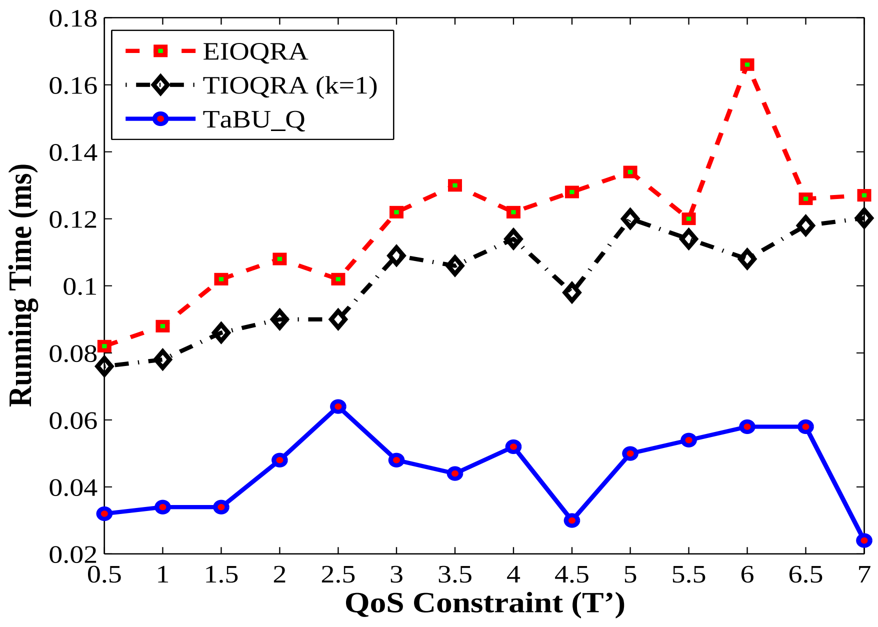


Figure 3-2: Running time vs QoS constraint for the lattice network with $N = 49$ under totally loose physical constraints.

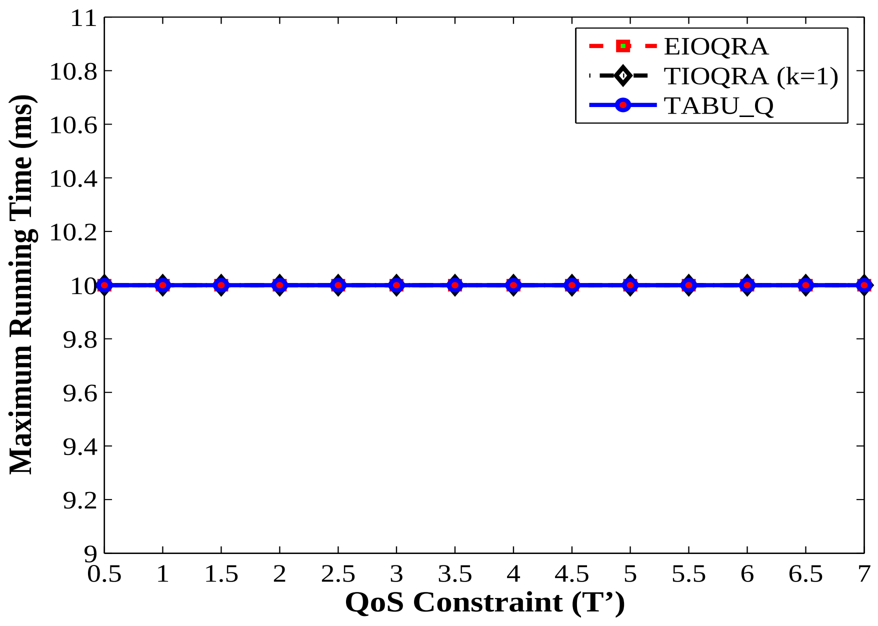


Figure 3-3: Maximum running time vs QoS constraint for the lattice network with $N = 49$ under totally loose physical constraints.

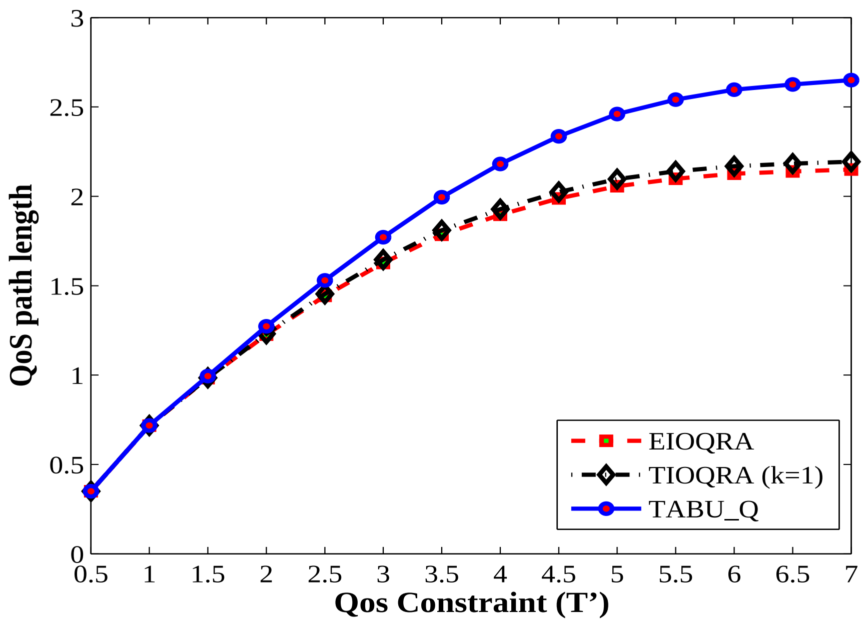


Figure 3-4: QoS path length vs QoS constraint for the lattice network with $N = 49$ under totally loose physical constraints.

processor has a resolution of $10ms$, and only $0ms$ and $10ms$ are obtained by the algorithms.

Figure 3-1 depicts that TABU_Q obtains a lower acceptance ratio and a larger path length compared to EIOQRA and TIOQRA. It can be observed from Figure 3-4 and Figure 3-2 that EIOQRA and TIOQRA obtain shorter path lengths at the cost of increased running time compared to TABU_Q. The difference between the algorithms stems from the fact that EIOQRA (TIOQRA) exhaustively explores all (most of) the possibilities in order to obtain the shortest feasible path; whereas, TABU_Q stops after performing a fixed number of iterations. The performance of EIOQRA slightly differs from that of TIOQRA in that EIOQRA provides a slightly better acceptance ratio and shorter path length. As shown in Figure 3-2, EIOQRA has a slightly larger running time than TIOQRA.

The simulations conducted on ARPANET and random networks with these algorithms also show similar behaviors as the lattice network. The results of these simulations are given in Apendix B-1.

3-3 Simulation under Totally Loose QoS Constraints

In this scenario, the QoS constraints are loosened so that no path will possibly violate them. This is done by setting $T_1 = T_2 = T' = N - 1$, where N is the number of nodes in the network. The performance of the algorithms is measured by gradually varying the physical constraints for different values of the total number of regenerators in the network (N_R). In all the experiments, the physical constraints are assigned the same value ($\Delta_1 = \Delta_2 = \Delta'$). All of our algorithms are tested on ARPANET, random networks, and lattice networks.

The QoS optimization algorithms have been tested on a lattice network with $N = 49$ by using $k = 1$ for the tunable heuristic algorithm (TIOQRA). Figure 3-5 shows the plot of the acceptance ratio vs the physical constraint (Δ') when $N_R = 8$. For each algorithm, acceptance ratio increases with increase in Δ'. For larger Δ', less number of paths fail to satisfy the physical constraint, thereby resulting in higher acceptance ratio.

The QoS path length plots are depicted in Figure 3-6. As can be observed from the figure, the path length curves increase with increase in Δ' up to a certain point. In this region, the physical constraint is tight. Thus, as Δ' is increased longer paths will be accepted because of the relief in the constraint resulting in larger average path length. After a certain point, the path length curves start to decline slowly. In this region, the physical constraints are not tight. Thus, increasing Δ' gives rise to increased freedom to optimize the QoS path length, thereby yielding slightly smaller path length. Due to the same reason mentioned in the previous scenario, the path lengths are randomly distributed up to the maximum value yielding standard deviation which is in the order of the average value at each data point. The figures reveal that EIOQRA and TIOQRA outperform TABU_Q in terms of acceptance ratio and the path length. As expected, EIOQRA achieves better results than TIOQRA in terms of these performance metrics.

The plot of the running time vs the physical constraint shown in Figure 3-7 is obtained for the exact algorithm (EIOQRA). As can be observed from the figure, the running time of EIOQRA becomes notably large in the region where the physical constraint is medium (neither tight nor loose). When Δ' is small, a (sub)path on average traverses only few nodes before it violates the physical constraint. Thus, a (sub)path requires regeneration at small intervals. But the probability of finding a regenerator in a small interval is less, resulting in smaller probability of regeneration. With less probability of regeneration, more paths are dropped and fewer paths are stored, thereby reducing the running time. For large Δ', a path can easily be found within short time. However, when Δ' is medium, the probability that a (sub)path traverses a regenerator before it violates the physical constraint is higher compared to when Δ' is small. This results in higher probability of regeneration, thereby decreasing the number of dropped paths. Furthermore, paths are not successfully found as easily as when Δ' is large. This combination leads to a scenario where more exhaustive search that stores larger number of paths at nodes is employed, resulting in long running time especially when the path cannot be found. A similar behavior is observed by repeating the simulation for $N_R = 16$. Compared to the case when $N_R = 8$, the long running time is observed for smaller values of Δ'. This reveals that the extent of the looseness or tightness of the physical constraint

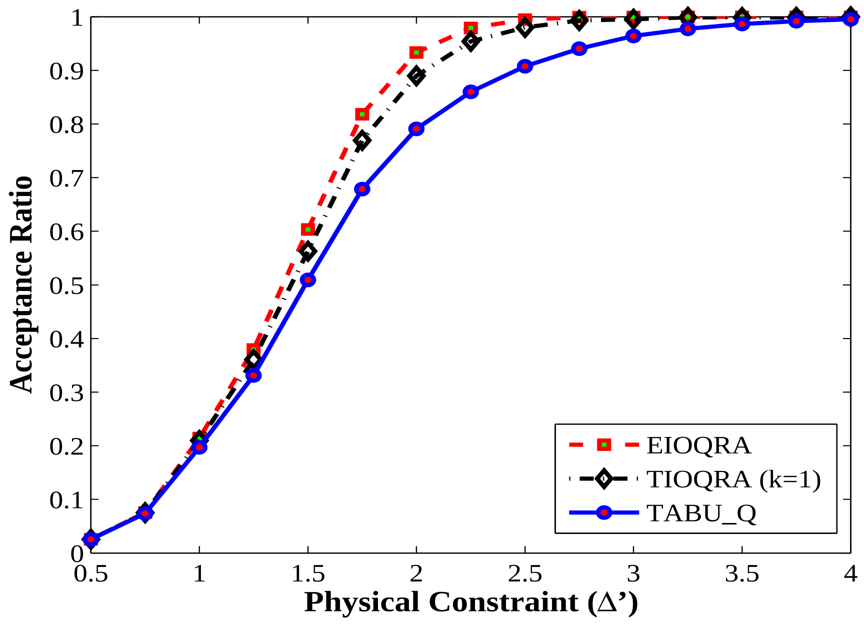


Figure 3-5: Acceptance ratio vs physical constraint for the lattice network with $N = 49$ and $N_R = 8$ under totally loose QoS constraint.

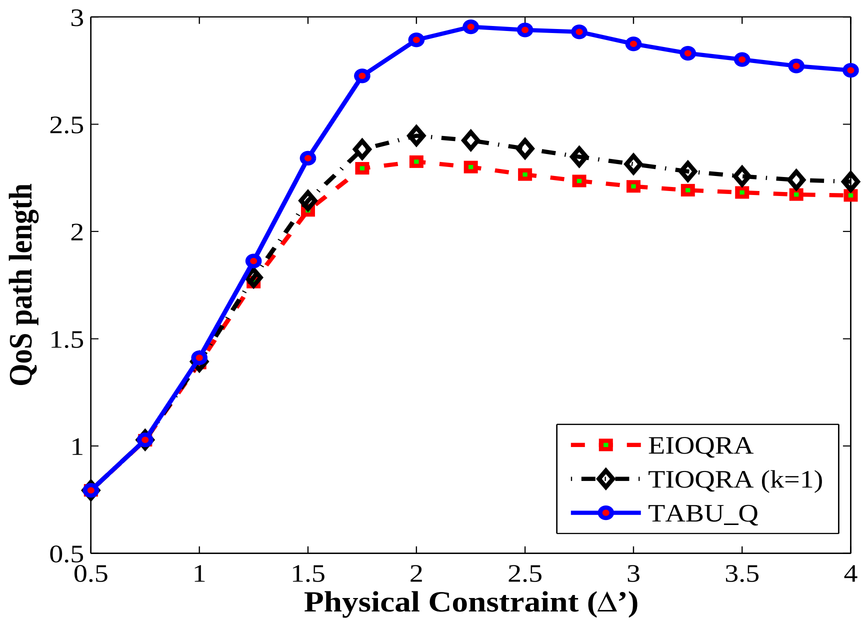


Figure 3-6: QoS path length vs physical constraint for the lattice network with $N = 49$ and $N_R = 8$ under totally loose QoS constraint.

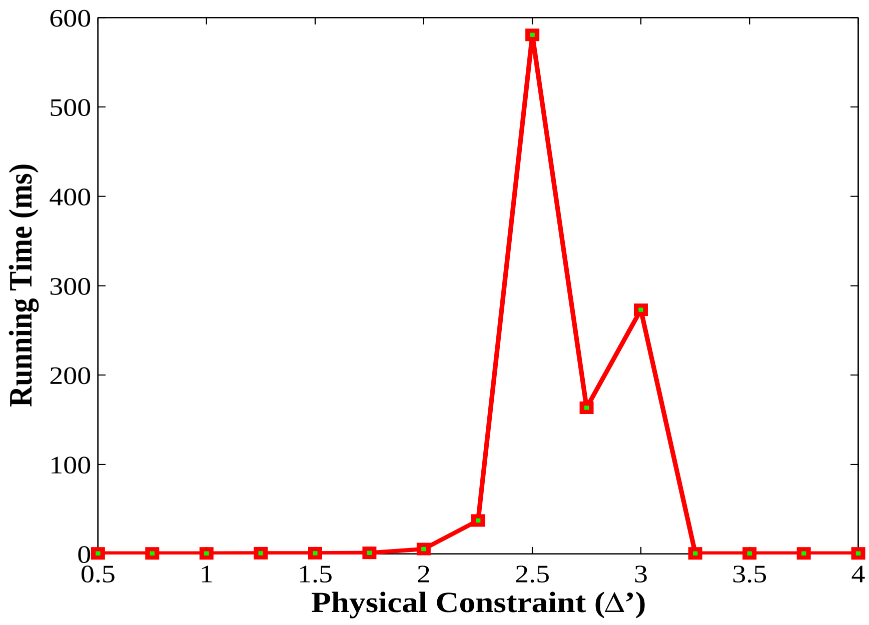


Figure 3-7: Running time vs physical constraint for EIOQRA on the lattice network with $N = 49$ and $N_R = 8$ under totally loose QoS constraint.

does not solely depend on the value of Δ', but also on N_R. Thus, when we say the physical impairment is tight, medium, or loose, we are actually referring to a combination of Δ' and N_R.

The plot of the running time vs the physical constraint for the three algorithms is depicted in Figure 3-8. In this figure, the running time of the exact algorithm in the medium-constrained region is intentionally truncated to closely observe the behavior of the algorithms in the other regions. In the tight-constrained and loose-constrained regions, EIOQRA has slightly higher running times compared to TIOQRA. As was the case with the previous results, TABU_Q has the smallest running time. A similar behavior is observed by repeating the simulation for $N_R = 16$. The maximum running time plot of EIOQRA is shown in Figure 3-9. As can be observed from the figure, large running times are observed in the medium-constrained region. In this region, the exact algorithm spends long time on unsuccessful searches due to the same reason discussed before. The maximum running time observed for TIOQRA and TABU_Q is 10ms at each data point.

The results of the simulations conducted on random networks also show similar behaviors as the lattice network, and are presented in Appendix B-2. Whereas for ARPANET, the simulation results show similar behavior with the lattice network but the large running time of EIOQRA in the medium-constrained region is missing. This is because ARPANET is a network with relatively smaller number of nodes and links

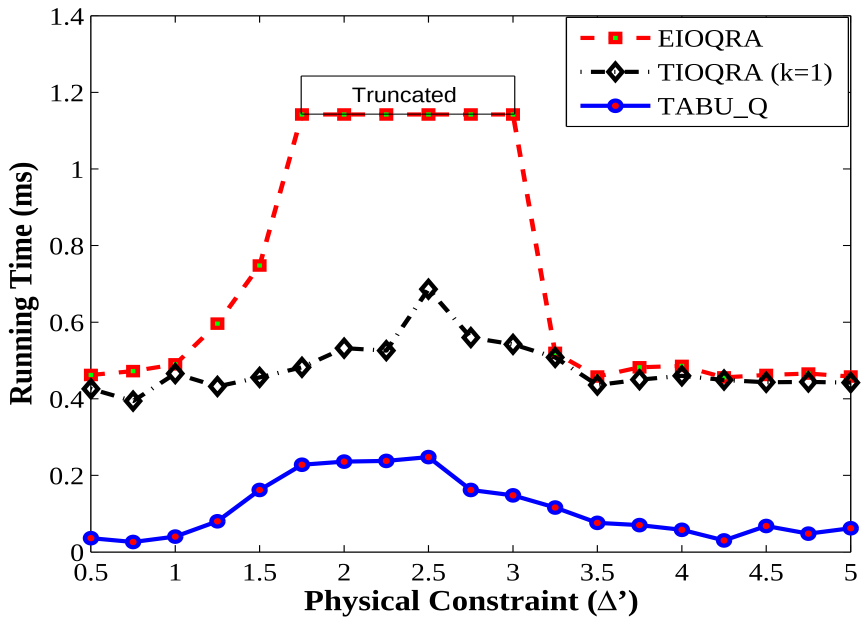


Figure 3-8: Running time vs physical constraint for the lattice network with $N = 49$ and $N_R = 8$ under totally loose QoS constraint.

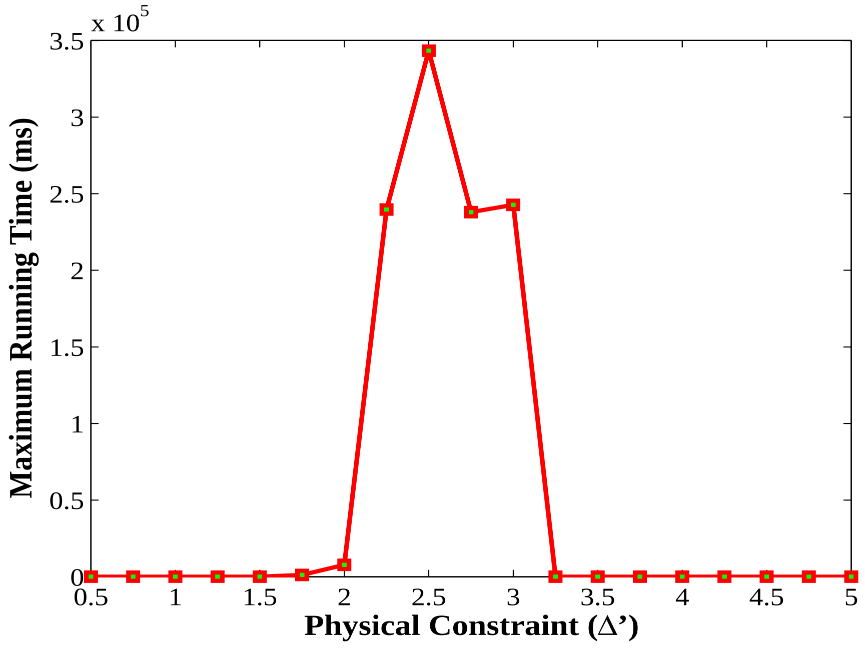


Figure 3-9: Maximum running time vs physical constraint for EIOQRA on the lattice network with $N = 49$ and $N_R = 8$ under totally loose QoS constraint.

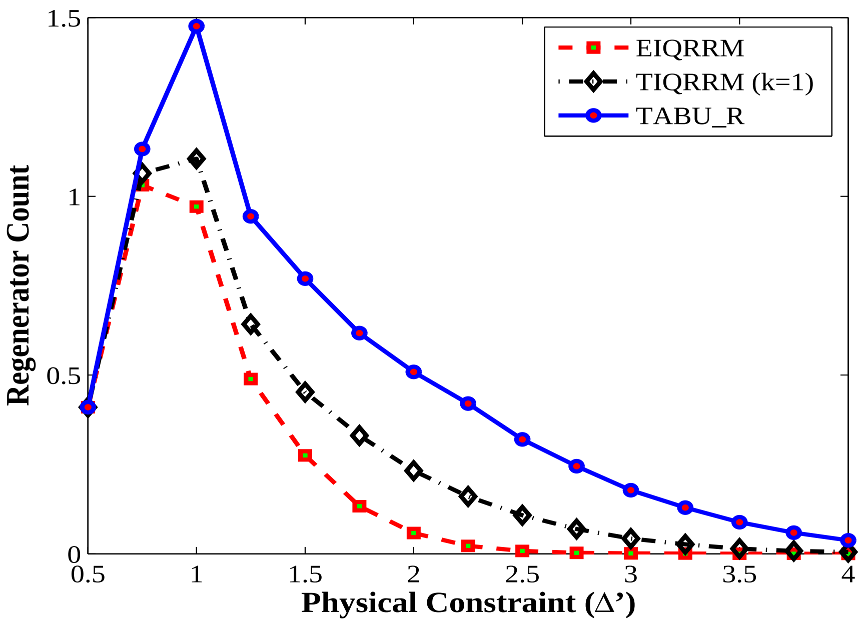


Figure 3-10: Regenerator-count vs physical constraint for the random networks with $N = 49$ and $R_N = 16$ under totally loose QoS constraint.

resulting in a relatively smaller search space.

Simulations are also performed with the regenerator-count minimization algorithms (EIQRRM, TABU_R, and TIQRRM) on the three network topologies. For the random network, the simulation is conducted with $N_R = 16$. Figure 3-10 shows the plot of the path regenerator-count vs the physical constraint for EIQRRM, TABU_R and TIQRRM. When Δ' is small, a (sub)path on average traverses few nodes before it violates the physical constraint. Thus, a (sub)path requires regeneration at small intervals; otherwise it will be dropped. Since the number/density of regenerators in the network is fixed, only shorter paths that require none or few regenerations are accepted resulting in small average path regenerator-count. However, when Δ' is increased to some extent, the path regenerator-count also increases until it reaches a maximum value before it starts to decline. In this scenario, Δ' is neither too small nor too large. A (sub)path now can traverse more number of nodes before it violates Δ'. The more nodes traversed, the more the chance to find regenerator node(s). This leads to a better chance of regeneration in times of need. Thus, paths that demand more regenerations have better chance to be accepted, thereby resulting in increased average path regenerator-count. However, when Δ' gets larger and larger, the average path regenerator-count declines since most of the paths are accepted without the need for regenerations at several nodes.

The regenerator-count of the individual connection requests is randomly distributed between zero and the maximum value because the source and the destination nodes are selected randomly. Consequently, the computed standard deviations at each data points are in the order of the average value. It can be observed that EIQRRM attains a smaller regenerator-count than TABU_R and TIQRRM. The acceptance ratio and the running time plots obtained for these algorithms have similar patterns with the results obtained for the corresponding QoS optimization algorithms presented earlier. In addition, the same simulation was repeated with $R_N = 8$ yielding results of similar patterns.

The results obtained from simulations on ARPANET and lattice network also show similar behaviors. These results are presented in Appendix B-2.

3-4 Simulation under Tight QoS Constraints

In this section, we present the simulation results obtained under the scenario where the QoS constraint is tight. After fixing the QoS constraint at certain tight value, the performances of the algorithms are measured by varying the number of regenerators in the network for a fixed value of the physical constraint. Simulations are conducted with each algorithm on all of the three network topologies. In all the simulations, $\Delta_1 = \Delta_2 = \Delta'$, and $T_1 = T_2 = T'$.

First, we present the simulation results obtained for the QoS optimization algorithms. On a random network with $N = 49$ and $\rho = 0.1$ the simulation is run by setting $\Delta' = 1$, $T' = 4$, and $k = 1$ for TIOQRA. The acceptance ratio plots of the algorithms are given in Figure 3-11. As the total number of regenerators in the network is increased, more paths are accepted because there will be more probability of regeneration. As can be observed from Figure 3-12, the running time of EIOQRA and TIOQRA increase with increase in the total number of regenerators in the network. This arises from the time spent by the INITIALIZE subroutine to compute the physical impairment lower bound vector $\overrightarrow{b^*}$. The time taken to compute $\overrightarrow{b^*}$ is linearly related to N_R because Dijkstra's algorithm is executed for each regenerator node in the network. The maximum running time plots are shown in Figure 3-13. For TIOQRA and TABU_Q, the maximum running time observed is $10ms$ at each data point. But for EIOQRA, a maximum running time of $3s$ is observed for larger values of R_N. Even though Δ' is small, for large values of N_R EIOQRA spends relatively larger time on unsuccessful path searches because there will be larger probability of regeneration that increases the search space. However, the search space is limited also by the tight QoS constraint. Therefore, the maximum running time is not as large as the values observed in the scenario where the QoS constraint is totally loose.

The QoS path length plots of the algorithms are depicted in Figure 3-14. As R_N is increased, the average path length also increases because longer paths can be accepted due to the increased probability of regeneration. But, after a certain point the average

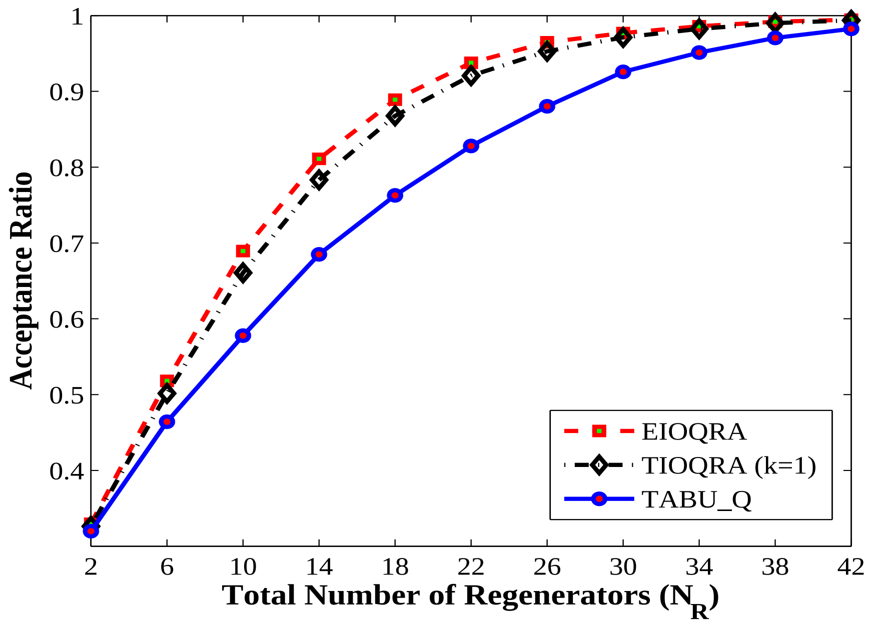


Figure 3-11: Acceptance ratio vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1$, and $T' = 4$

.

path length starts to decline slowly. As R_N increases beyond this point, the algorithms will entertain more freedom to optimize the QoS path length because the physical constraint is getting more and more loose. This results in decline in the average path length. The standard deviation at each data point is in the order of the average value due to the same reasons discussed in the previous scenarios.

Like in the previous scenarios, the results of the simulations presented in the figures show that EIOQRA and TIOQRA achieve better acceptance ratio and path length than TABU_Q. Whereas, TABU_Q yields the best running time. Additional simulations are also performed on ARPANET, random networks of $N = 100$ and $\rho = 0.05$, and lattice networks of $N = 49$ and $N = 100$. The results of these simulations also show similar patterns, and some of them are presented in Appendix B-3.

The regenerator-count minimization algorithms are also tested on all the three network topologies under tight QoS constraints. The results of these simulations show that the acceptance ratio and the running time of the algorithms exhibit similar behaviors with that of the corresponding QoS optimization algorithms. Figure 3-15 shows the plot of the path regenerator-count obtained from a simulation conducted on random networks of $N = 100$ and $\rho = 0.1$ with the parameters $\Delta' = 1$, $T' = 3$, and $k = 1$ for TIOQRA. As can be observed from the figure, EIQRRM attains the best path regenerator-count, whereas TABU_R attains the worst result for each value of N_R. Since Δ' is small, increasing N_R

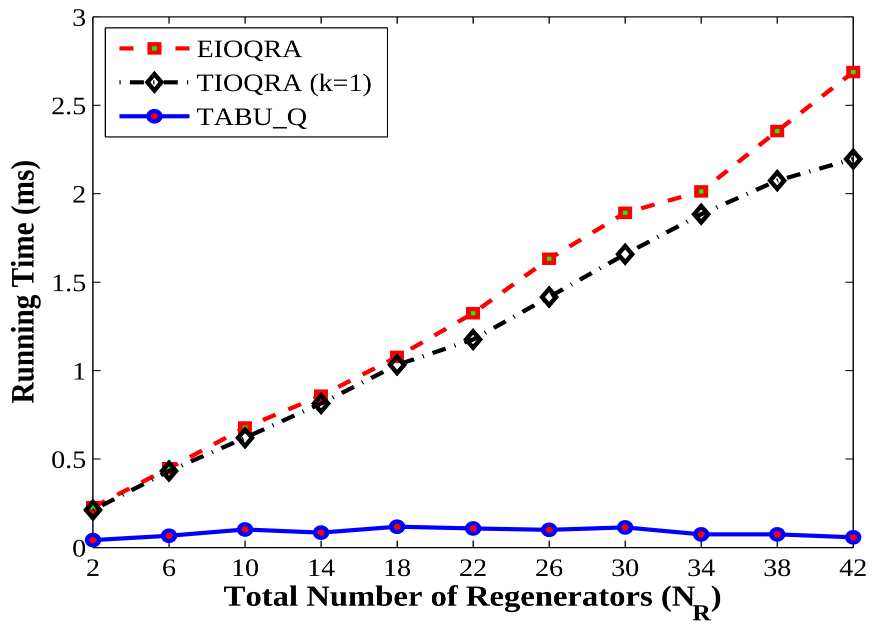


Figure 3-12: Running time vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1$, and $T' = 4$

.

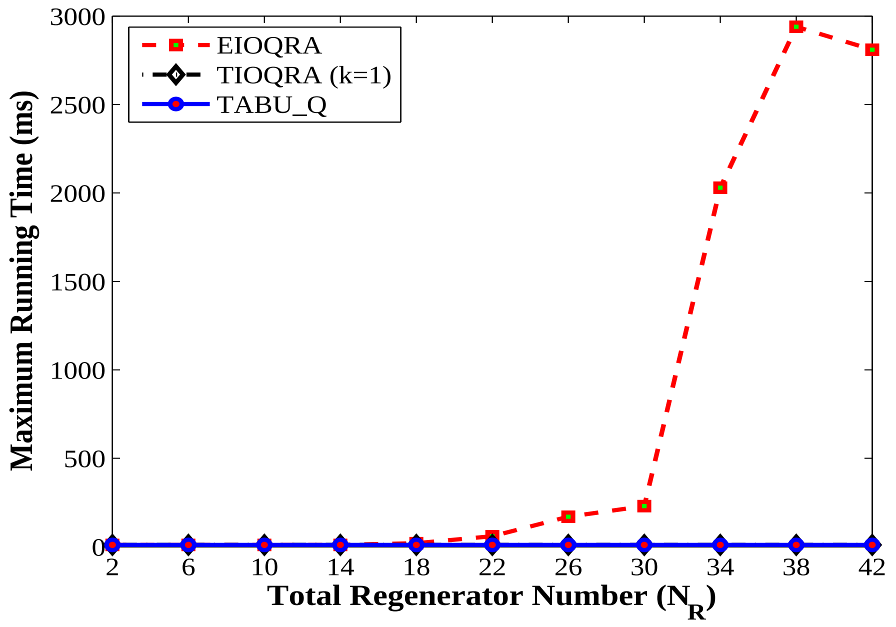


Figure 3-13: Maximum running time vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1$, and $T' = 4$

.

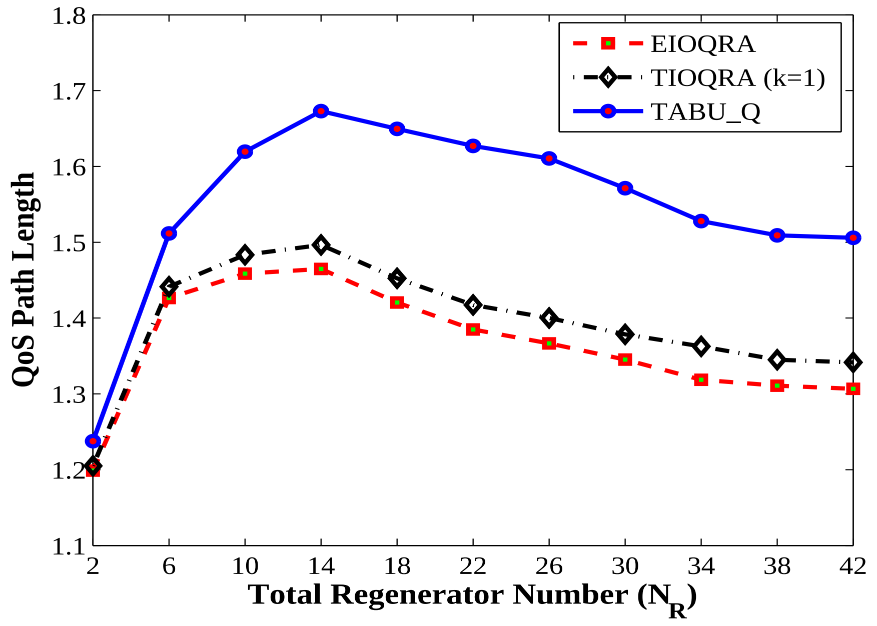


Figure 3-14: Qos path length vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1$, and $T' = 4$

.

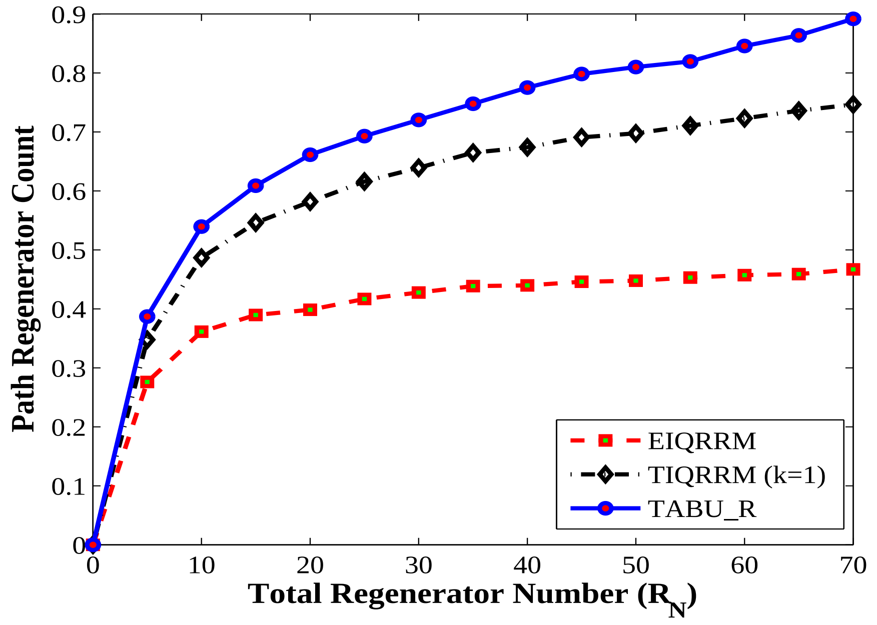


Figure 3-15: Path regenerator-count vs the total regenerator number for the random networks with $N = 100$, $\rho = 0.1$, $\Delta = 1$, and $T = 3$

.

leads to the acceptance of paths that demand more number of regenerations. As a result, the average path regenerator-count increases with increase in N_R. By the same reasonings mentioned in the previous scenarios, the standard deviations of the regenerator-count are in the order of the average values.

3-5 Simulation under Medium QoS Constraints

In this scenario, the QoS constraint is set at a relatively medium value. After fixing the physical constraints at a certain value, the performance of the algorithms are measured by varying the total number of regenerators in the network. We set $\Delta_1 = \Delta_2 = \Delta'$ and $T_1 = T_2 = T'$ in all the simulations.

A simulation was conducted with the QoS optimization algorithms on random networks of $N = 49$ and $\rho = 0.1$ with $\Delta' = 1.5$ and $T' = 6$. As shown in Figure 3-16, the acceptance ratio of the algorithms increases with increase in N_R, because less number of requests are blocked when N_R is large due to the increased probability of regeneration. The plot of the QoS path length of the algorithms is depicted in Figure 3-17. As was true in the previous scenarios, the path lengths are randomly distributed up to the maximum value, thereby attaining standard deviation which is in the order of the average value. The results show that EIOQRA attains the best acceptance ratio and path length, whereas TIOQRA achieves better results compared to TABU_Q.

As can be observed from Figure 3-18, the running time of EIOQRA suddenly increases to a large value in the region where the physical constraint is medium. This happens due to the same reason discussed in the scenario where the QoS constraint is totally loose. The peak value of the running time, however, is not as large as the peak value observed in the scenario where the QoS constraints are totally loose. Unlike the case of totally loose QoS constraints where no path violates the QoS constraints, in the case of medium Qos constraints the QoS look-ahead reduces the search space by dropping paths, thereby reducing the running time. Figure 3-19 shows the maximum running time plot of EIOQRA. The increase in the maximum running time can also be explained in the same way as the corresponding case in the scenario of totally loose QoS constraints. For TIOQRA and TABU_Q, a maximum running time of $10ms$ is observed.

The plot of the running time of the three algorithms is shown in Figure 3-20 where the running time of EIOQRA is truncated to observe the running time of the other algorithms closely. It can be observed that TABU_Q gives the best running time, whereas TIOQRA yields a better running time than EIOQRA. The simulation performed on lattice network also shows similar patterns, and is provided in Appendix B-4.

For the regenerator-count minimization algorithms, a simulation is performed on

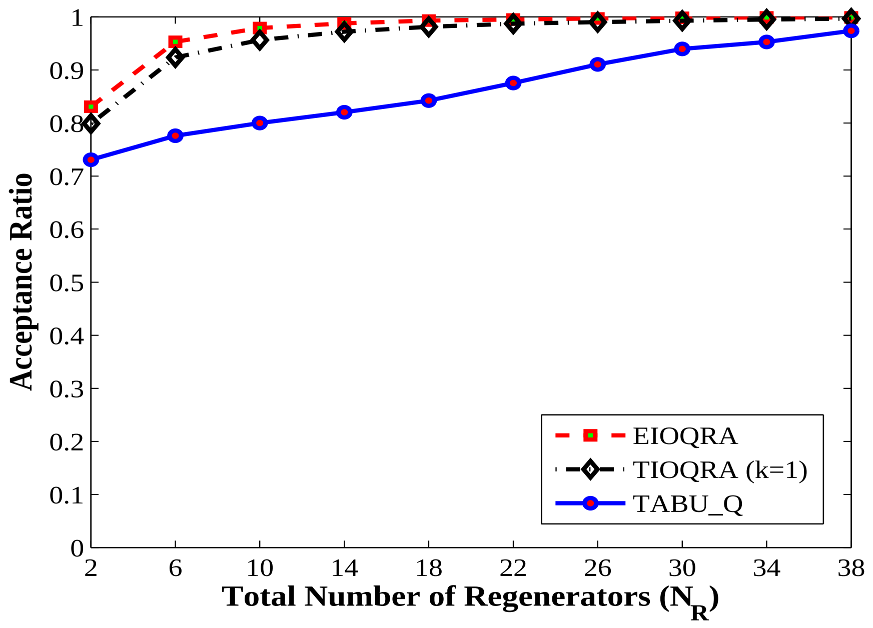


Figure 3-16: Acceptance ratio vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1.5$, and $T' = 6$

.

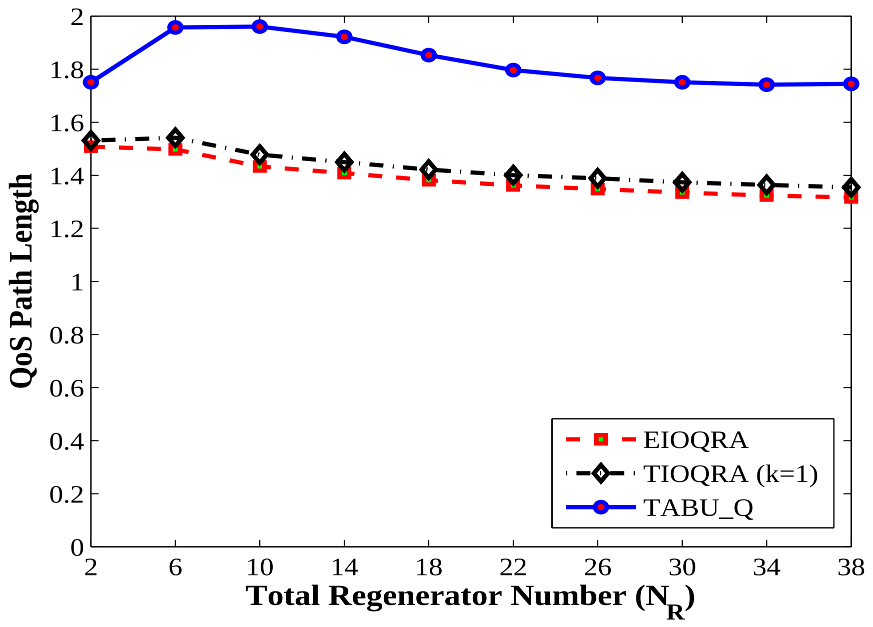


Figure 3-17: QoS path length vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1.5$, and $T' = 6$

.

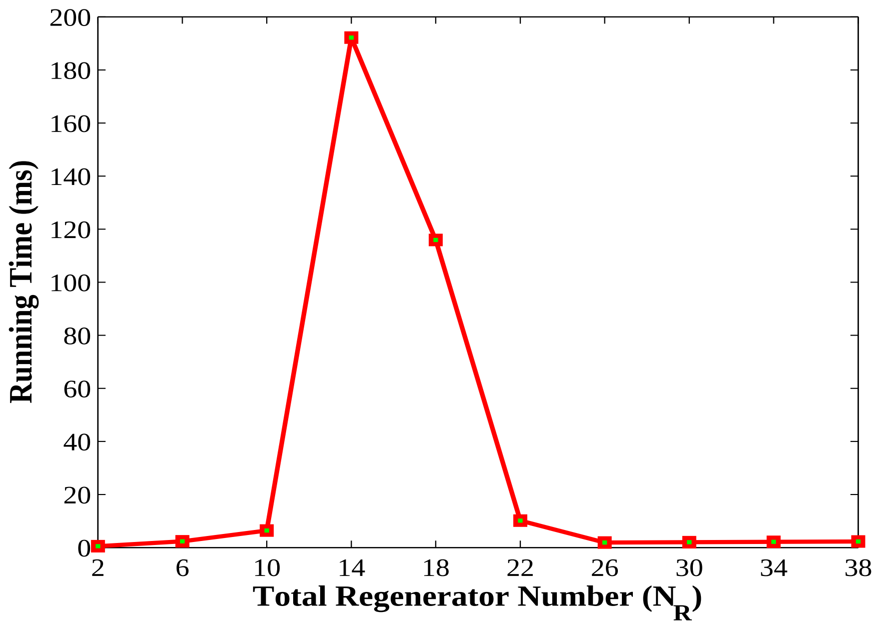


Figure 3-18: Running time vs the total regenerator number for EIOQRA on the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1.5$, and $T' = 6$

.

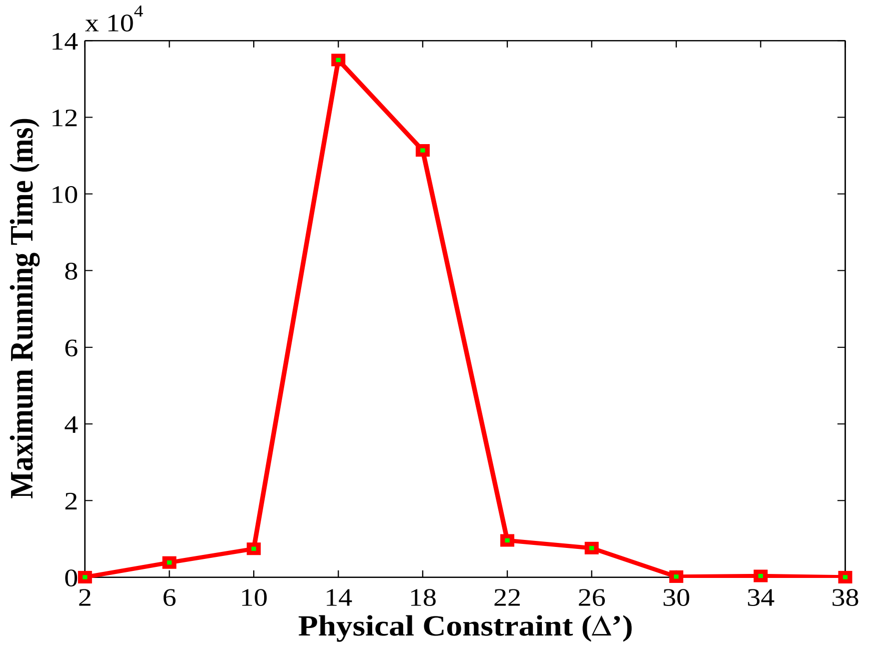


Figure 3-19: Maximum running time vs the total regenerator number for EIOQRA on the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1.5$, and $T' = 6$

.

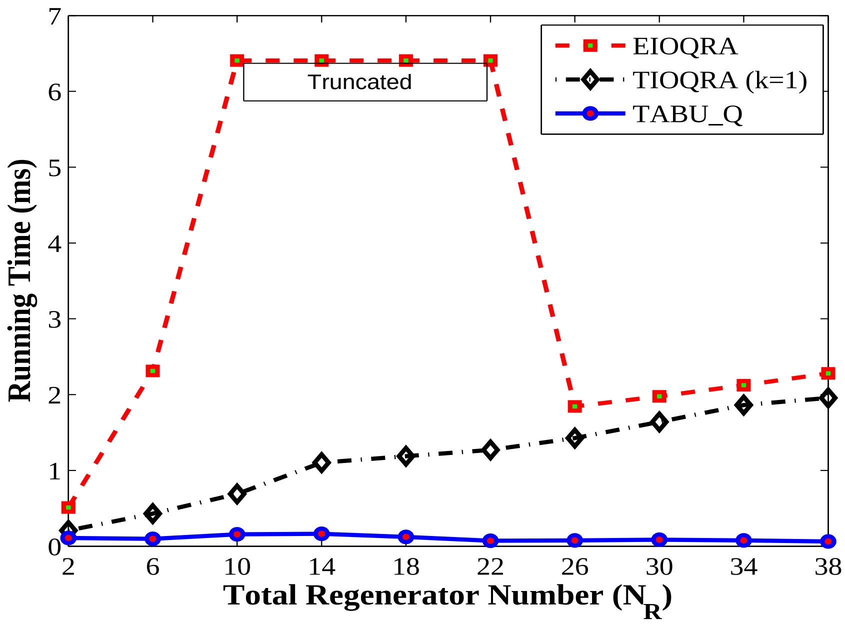


Figure 3-20: Running time vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1.5$, and $T' = 6$

.

random networks of $N = 49$ with $\Delta' = 1.5$ and $T' = 6$. The performance of these algorithms in terms of the running time and the acceptance ratio show similar behaviors with the performance of the corresponding QoS optimization algorithms. The plot of the path regenerator-count is shown in Figure 3-21. As revealed by the plots in the figure, EIQRRM shows the best performance, while TABU_R has the least performance in terms of the path regenerator-count. Like in the previous scenarios, the standard deviation of the regenerator-count is in the order of the average value revealing that the values are randomly distributed from zero to the maximum value. The simulation conducted on lattice network also shows similar patterns. The result of this simulation is provided in Appendix B-4.

The performances of the tunable algorithms depend on the number of paths that can be stored at a node (k). To study the effect of k on the performance of the algorithms, TIOQRA and TIQRRM have been simulated with three different values of k (1, 10, and 50) on random networks. The results of these simulations are shown in Appendix B-6. With $k = 50$ and $k = 10$, better performances are observed in terms of the acceptance ratio, the QoS path length, and the path regenerator-count than with $k = 1$. On the other hand, slightly lower running time is observed for $k = 1$ compared to the other values of k. It is also observed that the acceptance ratio, path length, and path regenerator-count obtained with $k = 50$ and $k = 10$ are very close to each other. Whereas, a slightly lower running time is achieved with $k = 10$. Further, the maximum running time observed for

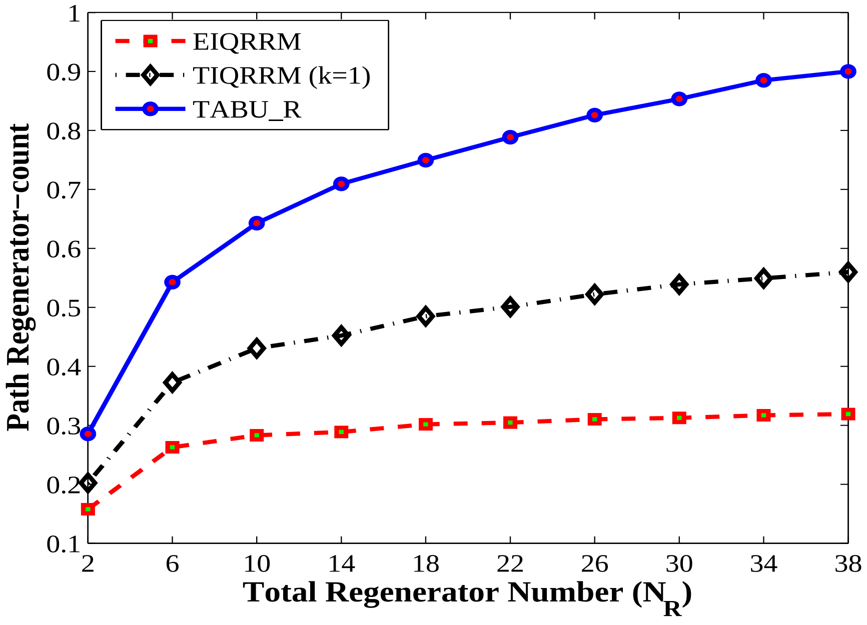


Figure 3-21: Path regenerator-count vs the total regenerator number for the random networks with $N = 49$, $\rho = 0.1$, $\Delta' = 1.5$, and $T' = 6$

.

$k = 1$ and $k = 10$ values is $10ms$; whereas, $30ms$ is observed with $k = 50$. Therefore, a k value around 10 can be a good choice to increase the performance in terms of the acceptance ratio, the path length, and the regenerator-count without a significant loose in the running time.

3-6 Simulation under Dynamic Traffic

In all the previous simulation scenarios, only a single connection request is considered per network. To see the performance of the algorithms under dynamic traffic, another set of simulations is performed in a different scenario. In this simulation scenario, a dynamic traffic of connection requests is generated. Each connection request is referred to as a flow. A flow is specified by a flow-id, a source, a destination, QoS constraints, physical constraints, arrival time, and departure time. Each fiber link in a network is associated with W wavelengths, two QoS link weights, and two physical impairment link weights. The link weights are assigned by a random function of uniform distribution. The regenerators are also randomly distributed in the network. For each flow, the source and destination nodes are randomly selected; and the arrival time and the departure time are random variables.

Whenever a flow arrives, the routing algorithm searches for a path from the source to the destination of the flow based on the requirements of the flow. If the algorithm fails to find a path that satisfies the constraints, the flow is rejected. But if a path that satisfies the constraints is found, the flow is accepted and the path is returned. Subsequently, the wavelength used in the path is reserved at every link along this path. Furthermore, the regenerators used along this path are reserved. The reserved resources cannot be used by another forthcoming flow until they are made available upon the departure of the flow that reserved them. Thus, both the list of unused wavelengths along each link and the list of available regenerators are modified accordingly. As a result, the state of the network evolves dynamically as new connections are established and the existing ones depart. The wavelength used along a path is selected by the best fit (BF) approach.

In the simulations, 10,000 flows are injected into the same network according to their arrival times. To compare the performance of the algorithms with respect to the running time, the QoS path length and the path regenerator-count, each of the algorithms should operate under the same scenario for each flow. To do so, each of the algorithms are applied one after the other (for each flow) on the same network, and the state of the network is updated by only one of them. On the other hand, the acceptance ratio of the algorithms is measured by separately applying the algorithms for the entire set of traffic on different copies of the same network and the throughput is measured.

The simulations are performed on random network, square lattice network, and ARPANET network with the exact and the tabu-search based heuristic algorithms. Some of the simulation results obtained are presented in Appendix B-5. In this simulation, the flows generated have a Poisson distribution. The inter-arrival time of the flows is at a rate of 4 per unit time. The flow duration is also exponentially distributed with a mean value of 4 unit times. The performance metrics are measured by varying the value of *regenerator density*. The regenerator density represents the probability that a node has a regenerator. The results of the simulation show that the exact algorithm outperforms the tabu-search based algorithm in terms of the path length and the path regenerator count. Yet, the tabu-search based algorithm attains a better running time than the exact algorithm. An interesting result obtained in this scenario is that the tabu-search based heuristic algorithm may achieve a better acceptance ratio than the exact algorithm (Figure B-39). To guarantee the optimal path solution for a flow, the exact algorithm consumes the “low-cost” links to set up the path. After reserving the resources for the current flow, the network may evolve to a state where the forthcoming flows are blocked because they can not be established with the remaining “high-cost” links. This situation leads to the possibility that the exact algorithm yields a smaller throughput than the tabu-search based algorithm.

Chapter 4

Discussion and Conclusion

In this thesis, we have solved the impairment-aware QoS routing (IQR) problem in translucent optical networks. In solving the IQR problem, we have also addressed two objectives: optimizing the end-to-end QoS metrics, and minimizing the number of regenerators used along a path. Our work differs from the previous works in that we incorporate both the physical impairments and the regenerator assignment in the path computation process.

In Chapter 2, our exact and heuristic algorithms to solve the IQR problem are presented for each of the two objectives. In chapter 3, we have analyzed the algorithms to study their behaviors in different scenarios.

4-1 Discussions

To solve the IQR problem with the QoS optimization objective, we have presented one exact and two heuristic algorithms. The exact algorithm (EIOQRA) inherits some concepts of SAMCRA [33]. The *k-shortest path* concept with unrestricted value of k, and a modified version of the *look-ahead* concept are used. One of the heuristic algorithms (TIOQRA) is a modification of EIOQRA obtained by merely restricting the value of k. The other heuristic algorithm (TABU_Q) uses the tabu-search approach.

We have simulated the algorithms under different scenarios on three different network topologies. Under the scenario where the physical constraints are extremely loose, on all the networks EIOQRA attains the largest acceptance ratio, and TIOQRA follows it with a small margin. Whereas, TABU_Q attains the least acceptance ratio. Another comparison metric we used is the length of the path selected in terms of the QoS metrics. With this comparison metric, EIOQRA shows the best performance in that it yields the shortest path length; whereas TABU_Q yields the longest path length. On the other hand, TABU_Q shows the best performance in terms of the running time, whereas EIOQRA shows the worst time performance. EIOQRA shows the best performance in terms of the acceptance ratio and the path length because it performs a more exhaustive search to obtain the optimal solution which costs longer running time. However, the running time of EIOQRA did not show any significant increase even when the QoS constraint is 'medium'. This is due to the fact that the look-ahead applied to the QoS metrics is effective in reducing the search space of possible paths, thereby decreasing the running time.

Under the scenario where the QoS constraints are extremely loose, EIOQRA shows the best performance in terms of acceptance ratio and path length; whereas TABU_Q shows the least performance in terms of these metrics. Meanwhile, TABU_Q shows the best performance in terms of the running time. The performance of TIOQRA is in between EIOQRA and TABU_Q in terms the three performance metrics. The running time of EIOQRA shows a remarkable increase in a specific situation when the total number of regenerators in the network is large and the physical constraint is neither loose nor tight (medium). In this situation, EIOQRA explores a significant number of possible paths in search of a solution when the path cannot be found, thereby increasing the running time. But this problem is not observed in the simulations performed on the realistic network (ARPANET).

When the QoS constraints are tight, EIOQRA shows the best performance in terms of the acceptance ratio and the path length. On the other hand, TABU_Q shows the best running time performance and the worst performance in terms of the acceptance ratio and the path length. The running time of EIOQRA is higher than that of TIOQRA only by a small margin because the tight QoS constraints help EIOQRA to highly reduce the search space of the possible paths. Likewise, TIOQRA follows EIOQRA with close margin in terms of the acceptance ratio and the path length.

In the scenario when the QoS constraints are medium, EIOQRA yields the best performance in terms of the acceptance ratio and the QoS path length; whereas, TABU_Q demonstrates the best running time performance. As was true with the other scenarios, TIOQRA exhibits better performance than TABU_Q in terms of the acceptance ratio and the QoS path length; and its running time is better than that of EIOQRA. EIOQRA spends long time on unsuccessful path search when the number of regenerators is large and the physical constraint is medium. But the increase in the running time of EIOQRA is less serious compared to the increase observed when the QoS is totaly loose because the search space is reduced by the QoS constraints.

We have also presented one exact and two heuristic algorithms that solve the IQR problem with the objective of minimizing the regenerator-count of a path. The algorithms are EIQRRM, TIQRRM, and TABU_R which are obtained by making some modifications on EIOQRA, TIOQRA, and TABU_Q, respectively. These algorithms are also tested under different scenarios, and their performance is measured in terms of the acceptance ratio, the running time, and the number of regenerators used along a path. In terms of the acceptance ratio and the running time, the algorithms show similar performance with their corresponding QoS optimization algorithms under all the scenarios we have considered. Under all the scenarios, in terms of the path regenerator count, the exact algorithm (EIQRRM) shows the best performance; whereas TABU_R shows the least performance.

Generally, the tabu-search based heuristic algorithms (TABU_Q and TABU_R) show the best performance in terms of the running time because they stop the path search after performing a fixed number of iterations. On the other hand, the exact algorithms (EIOQRA and EIQRRM) achieve the best performance in terms of the acceptance ratio, the path length, and the path regenerator-count because they perform exhaustive search to find the optimal solution which inevitably results in a longer running time. Whereas, the tunable heuristic algorithms (TIOQRA and TIQRRM) show intermediate performance in terms of all the performance metrics.

4-2 Conclusions

Each one of the algorithms proposed in this thesis has its own charm that makes it preferable over the others. Thus, the choice of a specific algorithm depends on the performance metric of our interest. The exact algorithms are preferable when exactness is the priority. However, the tabu search based algorithms are the best choice when the algorithm running time is exceptionally important. On the other hand, the tunable heuristic algorithms are attractive when a trade-off is made between the acceptance ratio and the algorithm running time.

4-3 Future Work

The results of our work are encouraging and there are potentially more works that can be done to expand/improve the algorithms. Therefore, we suggest the following possible directions of future work.

- In our routing algorithms, we did not consider path protection upon failure of a network element. However, in practice the failure of one network element can cause the failure of several optical channels, thereby leading to a huge data loss. Thus, a backup path is required for each optical channel in order to resume the connection immediately upon failure of the primary path. Therefore, we suggest a possible expansion work that incorporates path protection.
- When multiple optical channels simultaneously traverse the same link, they may interfere with each other. This interference is caused by the non-linear impairments. Thus, we suggest a future work that expands our algorithms to incorporate the effect of non-linear impairments in a dynamic environment.

Bibliography

[1] S. Azodolmolky, M. Klinkowski, E. Marin, D. Careglio, J. Pareta, and I. Tomkos, "A survey on Physical Layer Impairments Aware Routing and Wavelength Assignment Algorithms in Optical Networks," *Computer Networks,* vol. 53, no. 7, pp. 926-944, May 2009.

[2] J.M. Simmons, "On Determining the Optimal Optical Reach for a Longhaul Network," *Journal of Lightwave Technology,* vol. 23, no. 3, pp. 1039-1048, March 2005.

[3] S. Al Zahr, M. Gagnaire, N. Puech, and M. Koubaa, "Physical Layer Impairments in WDM Core Networks: a Comparison between a North-American Backbone and a Pan-European Backbone, *Proc. of the Int. Conf. on Broadband Networks,* vol. 2, pp. 1258-1263, October 2005.

[4] S. Sygletos, I. Tomkos, and J. Leuthold, "Technological Challenges on the Road Toward Transparent Networking," *Journal of Optical Networking,* vol. 7, no.4, pp. 321-350, April 2008.

[5] A.G. Strieegler, M. Meissner, K. Cvecek, K. Sponsel, G. Leuchs, and B. Schmauss, "NOLM-Based RZ-DPSK Signal Regeneration," *IEEE Photonics Technology Letters,* vol. 17, no. 3, pp. 639-641, March 2005.

[6] Y.K. Huang, L. Xu, I. Glesk, V. Baby, B. Li, and P.R. Prucnal, "Simultaneous All-Optical 3R Regeneration of Multiple WDM Channels," *Proc. of 18th annual meeting of the IEEE LEOS,* pp. 135-136, October 2005.

[7] G. Shen, and R.S. Tucker, "Translucent Optical Networks the Way Forward," *IEEE Communications Magazine,* vol. 45, no. 2, pp. 48-54, February 2007.

[8] A.L. Chiu, L. Guangzhi, and H. Dah-Min, "New Problems on Wavelength Assignment in ULH Networks," *Proc. of OFC/NFOEC,* March 2006.

[9] H. Zang, J. P. Jue, and B. Mukherjee, "A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks," *Optical Networks,* vol. 1, no. 1, pp. 47-60, January 2000.

[10] I. Chlamtac, A. Ganz, and G. Karmi, "Lightpath Communications: an Approach to High Bandwidth Optical WANs," *IEEE Transactions on Communications,* vol. 40, no. 7, pp. 11711182, July 1992.

[11] C. Siva Ram Murthy, and G. Mohan, *WDM Optical Networks: Concepts, Design, and Algorithms,* Prentice Hall PTR, 2002.

[12] B. Ramamurthy and B. Mukherjee, "Wavelength Conversion in WDM Networking," *IEEE Journal Selected Areas in Communications,* vol. 16, no. 7, pp. 1061-1073, September 1998.

[13] V. Sharma and E. A. Varvarigos, "Limited Wavelength Translation in All-Optical WDM Mesh Networks," *Proc. of INFOCOM,* vol. 2, pp. 893-901, March 1999.

[14] Y. Xin, G. N. Rouskas, and H. G. Perros, *On the Design of MPλS Networks,* Technical Report TR-01-07, North Carolina State University, July 2001.

[15] R. Dutta and G. N. Rouskas, "A Survey of Virtual Topology Design Algorithms for Wavelength Routed Optical Networks," *Optical Networks,* vol. 1, no. 1, pp. 73-89, January 2000.

[16] M. Ali Ezzahdi, S. Al Zahr, M. Koubaa, N. Puech, and M. Gagnaire, "LERP: a Quality of Transmission Dependent Heuristic for Routing and Wavelength Assignment in Hybrid WDM Networks," *Proc. of ICCCN,* pp. 125-136, October 2006.

[17] R. Cardillo, V. Curri, and M. Mellia., "Considering Transmission Impairments in Wavelength Routed Networks," *Proc. of ONDM,* pp. 421-429, February 2005.

[18] H. Yurong, J.P. Heritage, and B. Mukherjee, "Connection Provisioning with Transmission Impairment Consideration in Optical WDM Networks with High-Speed Channels," *Journal of Lightwave Technology,* vol. 23, no. 3, pp. 982-993, March 2005.

[19] J. He, M. Brandt-Pearce, Y. Pointurier, and S. Subramaniam, "QoT-Aware Routing in Impairment-Constrained Optical Networks," *Proc. of IEEE GLOBECOM,* pp. 2269-2274, November 2007.

[20] J. Strand, A. L. Chiu, and R. Tkach, "Issues for Routing in Optical Layer," *IEEE Communications,* vol.39, no. 2, pp. 81-96, February 2001.

[21] M. Farahmand, D. Awduche, S. Tibuleac, and D. Atlas, "Characterization and Representation of Impairments for Routing and Path Control in All-Optical Networks," *Proc. of NFOEC,* September 2002.

[22] B. Ramamurthy, D. Datta, H. Feng, J.P. Heritage, and B. Mukherjee, “Impact of Transmission Impairments on the Teletraffic Performance of Wavelength-Routed Optical Networks,” *Journal of Lightwave Technology*, vol. 17, no. 10, pp. 1713-1723, October 1999.

[23] J. Strand, and A. Chiu, “Impairments and Other Constraints on Optical Layer Routing,” *RFC4054,* May 2005.

[24] Y. Huang, W. Wen, J. P. Heritage, and B. Mukherjee, “Signal-Quality Consideration for Dynamic Connection Provisioning in All-Optical Wavelength-Routed Networks,” *Proc. of SPIE (OptiComm),* vol. 5285, pp. 163-173, 2003.

[25] G.P. Agrawal, *Nonlinear Fiber Optics, third ed.,* Academic Press, 2001.

[26] A. Marsden, A. Maruta, and K. Kitayama, “Routing and Wavelength Assignment Encompassing FWM in WDM Lightpath Networks,” *Proc. of IFIP ONDM”,* pp. 1-6, March 2008.

[27] X. Yang, L. Shen, and B. Ramamurthy, “Survivable Lightpath Provisioning in WDM Mesh Networks Under Shared Path Protection and Signal Quality Constraints,” *Journal Lightwave Technology,* vol. 23, no. 4, pp. 1556- 1567, 2005.

[28] R. Ramaswami, and K.N. Sivarajan, *Optical Networks: A Practical Perspective,* Morgan Kaufmann, California, 1998.

[29] C. Vijaya Saradhi, and C. Siva Ram Murthy, “Routing Differentiated Reliable Connections in WDM Optical Networks,” *Optical Networks Magazine*, vol. 3, no. 3, pp. 50-67, 2002.

[30] S. Ramamurthy and B. Mukherejee, “Survivable WDM Mesh Networks, Part I-Restoration,” *Proc. of IEEE INFOCOM,* vol. 2, pp. 744-51, March 1999.

[31] C. Xin, Y. Ye, S. Dixit, and C. Qiao, “A Joint Lightpath Routing Approach in Survivable Optical Networks,” *Proc. of SPIE Asia-Pacific Optical and Wireless Communications,* vol. 4585, pp. 139-146, November 2001.

[32] K. Wu, L. Valcarenghi, and A. Fumagalli, “Restoration Schemes with Differentiated Reliability,” *Proc. of IEEE ICC.,* vol. 3, pp. 1968-1972, May 2003.

[33] P. Van Mieghem and F.A. Kuipers, “Concepts of Exact Quality of Service Algorithms,” *IEEE/ACM Transaction on Networking,* vol. 12, no. 5, pp. 851-864, October 2004.

[34] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem, “An Overview of Constraint-Based Path Selection Algorithms for QoS Routing,” *IEEE Communication Magazine,* vol. 40, pp. 5055, December 2002.

[35] L. Ling and A.K. Somani, “Dynamic Wvelength Routing using Congestion and Neighbourhood Information,” *IEEE/ACM Transaction on Networking,* vol. 7, no. 5, pp. 779-786, October 1999.

[36] A. Mokhtar and M. Azizoglu, “Adaptive Wavelength Routing in All Optical Networks,” *IEEE/ACM Transaction on Networking,* vol. 6, no. 2, pp. 197-206, April 1998.

[37] S. Pachnicke, T. Paschenda, and P.M. Krummrich. “Physical Impairment Based Regenerator Placement and Routing in Translucent Optical Networks,” *Proc. of OFC/NFOEC,* February 2008.

[38] M. Ali Ezzahdi, S. Al Zahr, M. Koubaa, N. Puech, and M. Gagnaire, “LERP: a Quality of Tansmission Dependent Heuristic for Routing and Wavelength Assignment in Hybrid WDM Networks,” *Proc. of ICCCN,* pp. 125-136, October 2006.

[39] A. Jukan, and G. Franzl, “Constraint-Based Path Selection Methods for on Demand Provisioning in WDM Networks, *in: Proceedings of IEEE INFOCOM*, vol. 2, pp. 827-836, June 2002.

[40] H.A. Pereira, D.A.R. Chaves, C.J.A. Bastos-Filho, and J.F. Martins-Filho, “Impact of Physical Layer Impairments in All-Optical Networks, *Proc. of Microwave SBMO/IEEE MTT-S International IMOC,* pp. 536-541, November 2007.

[41] T. Deng, and S. Subramaniam, “Adaptive QoS Routing in Dynamic Wavelength-Routed Optical Networks,” *2nd Int. Conf. on Broadband Networks,* vol. 1, pp. 184-193, October 2005.

[42] J. He, M. Brandt-Pearce, Y. Pointurier, and S. Subramaniam, “QoT-Aware Routing in Impairment-Constrained Optical Networks,” *Proc. of IEEE GLOBECOM,* pp. 2269-2274, November 2007.

[43] G. Markidis, S. Sygletos, A. Tzanakaki, and I. Tomkos, “Impairment Aware Based Routing and Wavelength Assignment in Transparent Long Haul Networks,” *Proc. of IFIP ONDM,* vol. 4534, pp. 48-57, May 2007.

[44] F.Glover, “Tabu Search I,” *ORSA Journal on Computing,* vol. 1, no. 3, pp. 190-206, 1989.

[45] X. Yang, and B. Ramamurthy, “Dynamic Routing in Translucent WDM optical Networks the Intradomain Case, *Journal of Lightwave Technology,* vol. 23, no. 3, pp. 955-71, March 2005.

[46] J. He, M. Brandt-Pearce, Y. Pointurier, and S. Subramaniam, “Adaptive Wavelength Assignment Using Wavelength Spectrum Separation for Distributed Optical Networks, *IEEE International Conference on Communication,* pp. 24-28, Jun. 2007.

[47] E. Salvadori, Y. Ye, A. Zanardi, H. Woesner, M. Carcagni, G. Galimberti, G.Martinelli, A. Tanzi, and D. La Fauci, “Signalling-Based Architectures for Impairmentaware Lightpath Set-up in GMPLS Networks, *Proc. of IEEE GLOBECOM,* pp. 2263-2268, November 2007.

[48] N. Skorin-Kapov, “Heuristic Algorithm for the Routing and Wavelength Assignment of Scheduled Lightpath Demands in Optical Networks,” *IEEE Journal on Selected Areas of Communications,* vol. 24, pp. 2-15, August 2006.

[49] W. Yang. “Optimal and Heuristic Algorithms for Quality-of-Service Routing with Multiple Constraints,” *Performance Evaluation,* vol. 57, no. 3, pp. 261-278, 2004.

Appendix A

Network Topologies

A-1 The ARPANET Network

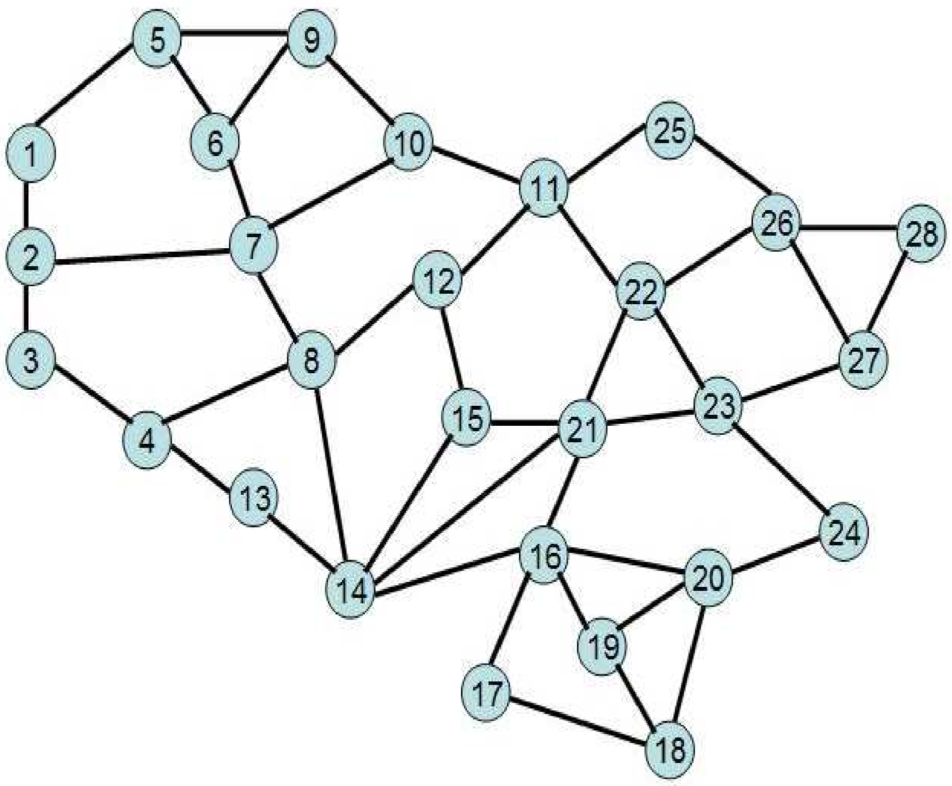


Figure A-1: The ARPANET network.

A-2 A Lattice Network

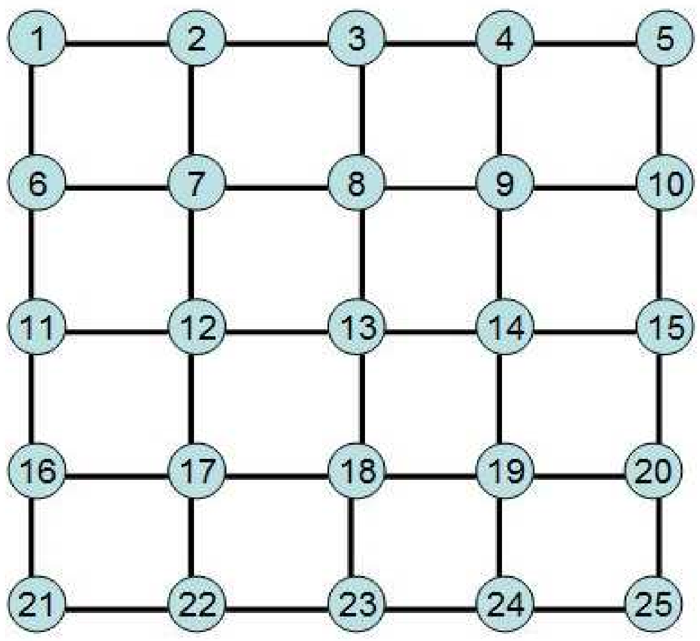


Figure A-2: A lattice network.

Appendix B

Additional Results

B-1 Totally loose physical constraints

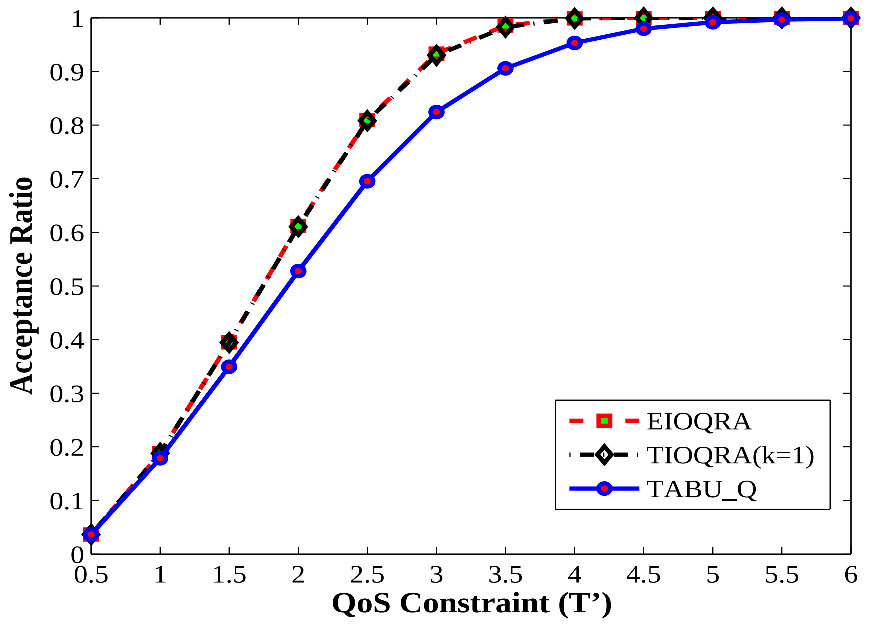


Figure B-1: Acceptance ratio vs the QoS constraint for ARPANET network.

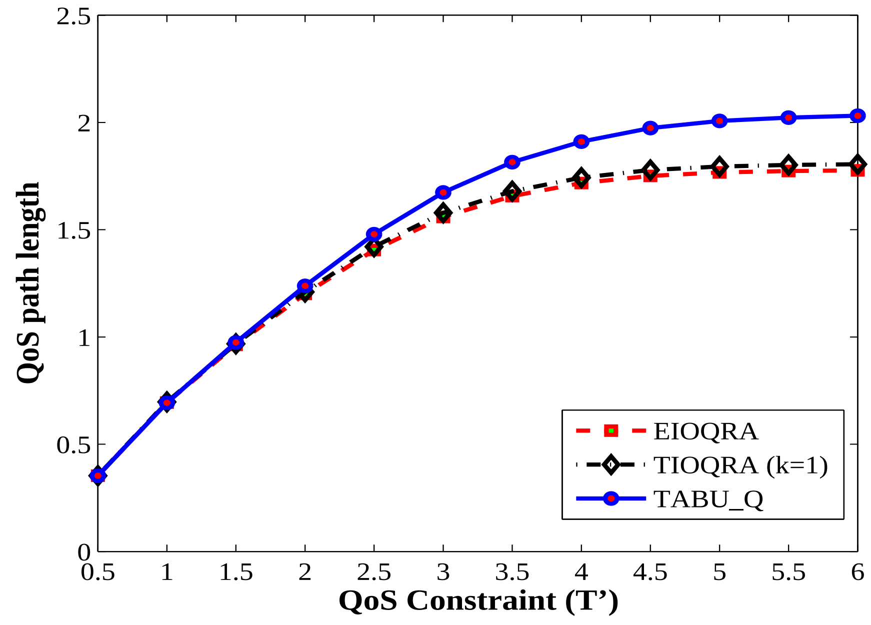


Figure B-2: QoS path length vs the QoS constraint for ARPANET network.

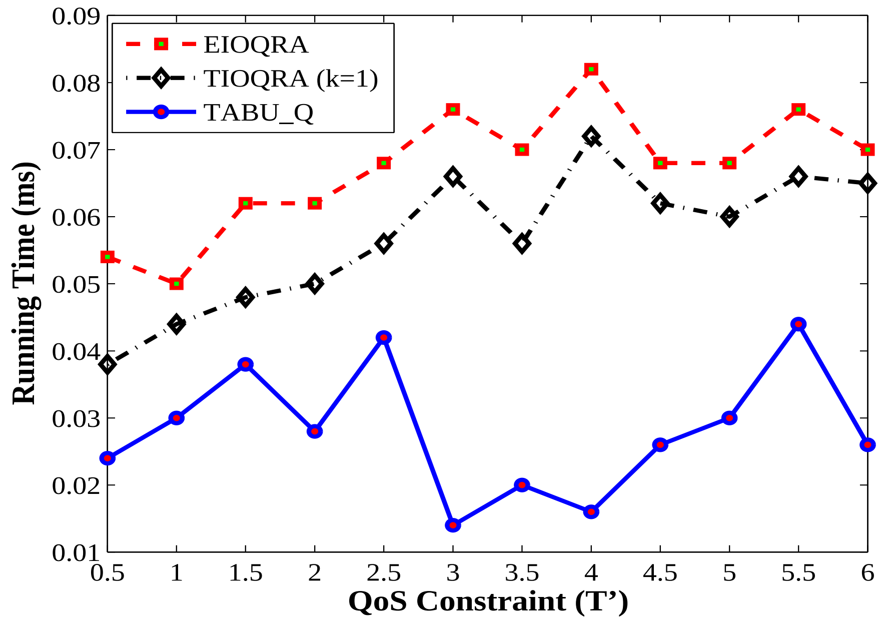


Figure B-3: Running time vs the QoS constraint for ARPANET network.

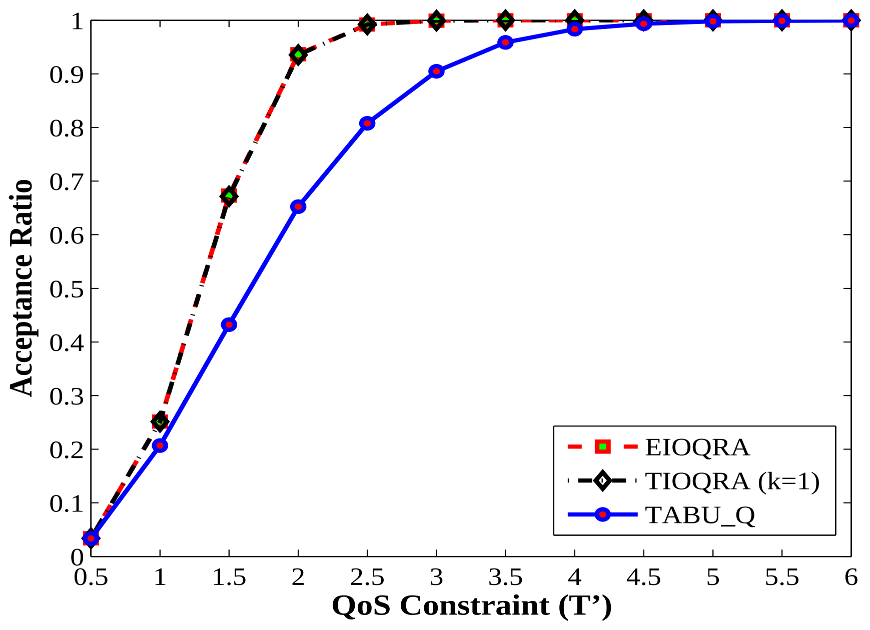


Figure B-4: Acceptance ratio vs the QoS constraint for random networks with $N = 49$.

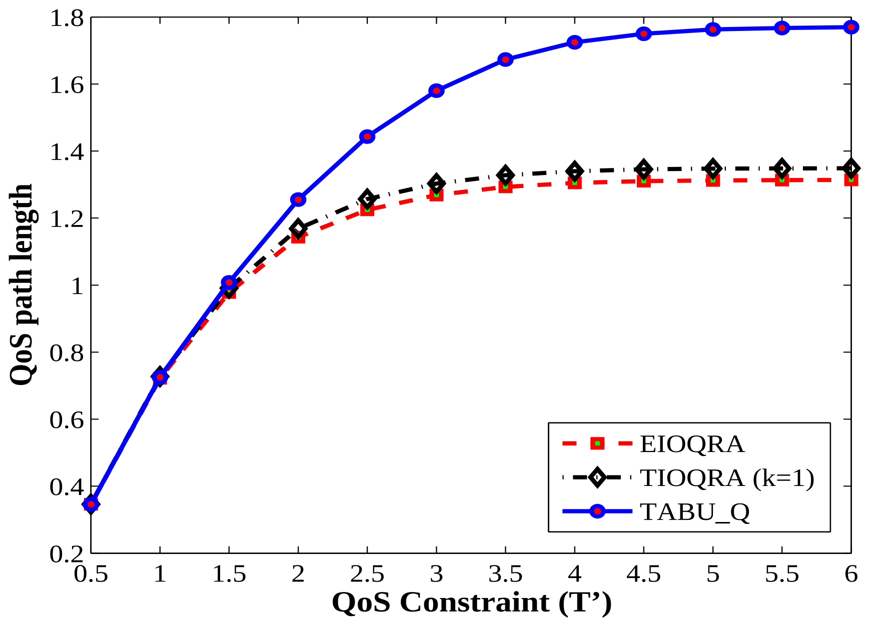


Figure B-5: QoS path length vs the QoS constraint for random networks with $N = 49$.

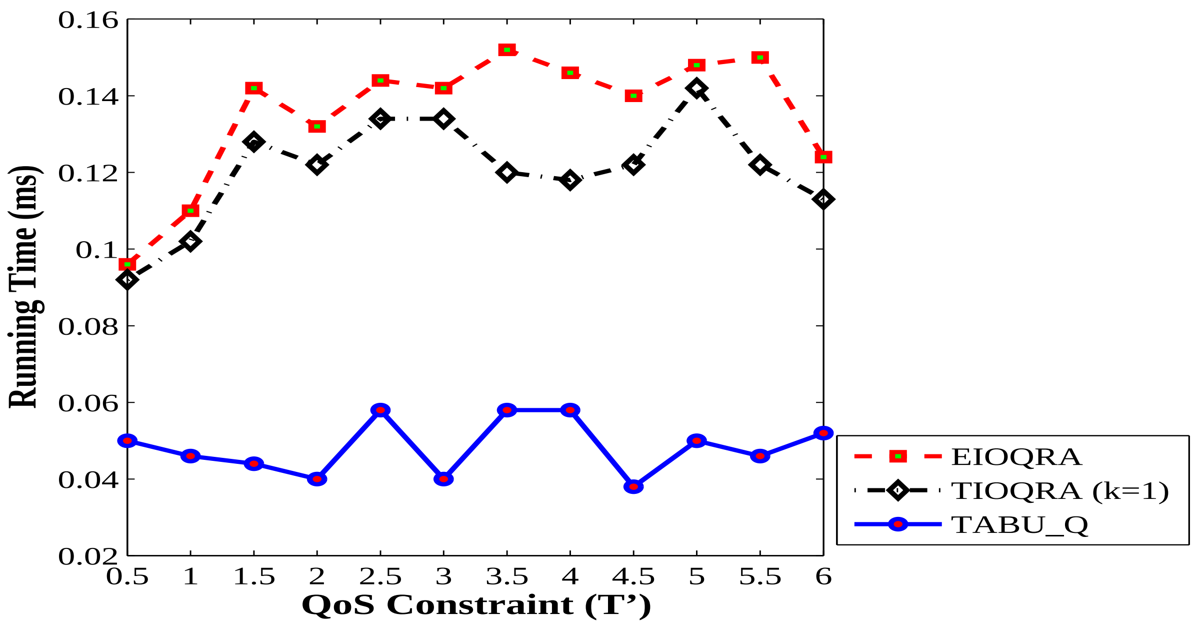


Figure B-6: Running time vs the QoS constraint for random networks with $N = 49$.

B-2 Totally loose Qos constraints

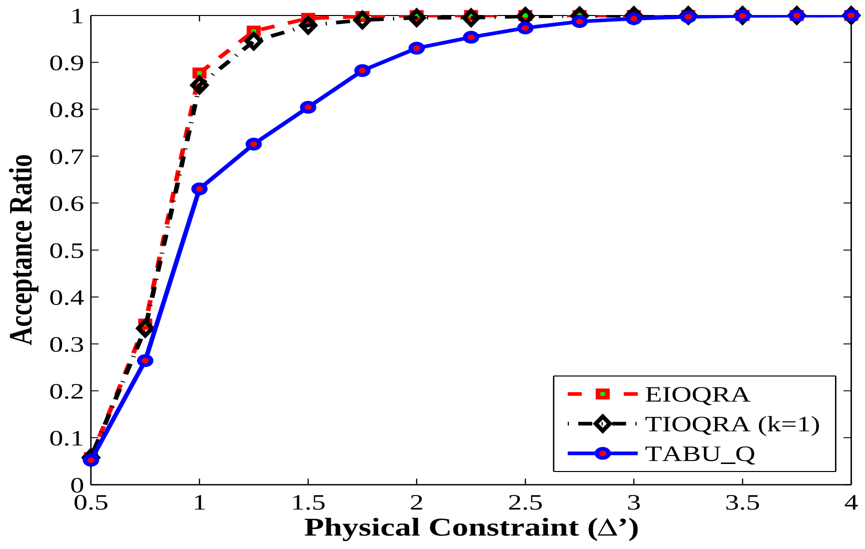


Figure B-7: Acceptance ratio vs physical constraint for the random networks with $N = 49$ and $N_R = 16$.

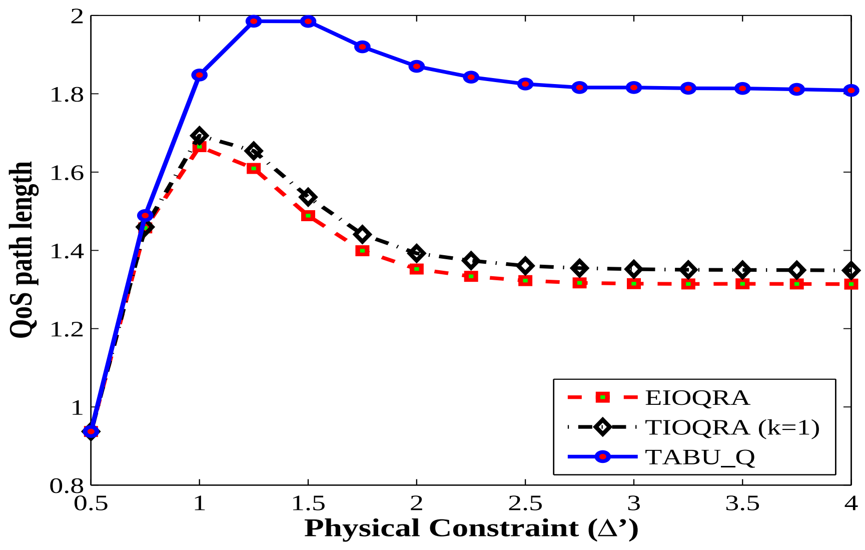


Figure B-8: QoS path length vs physical constraint for the random networks with $N = 49$ and $N_R = 8$.

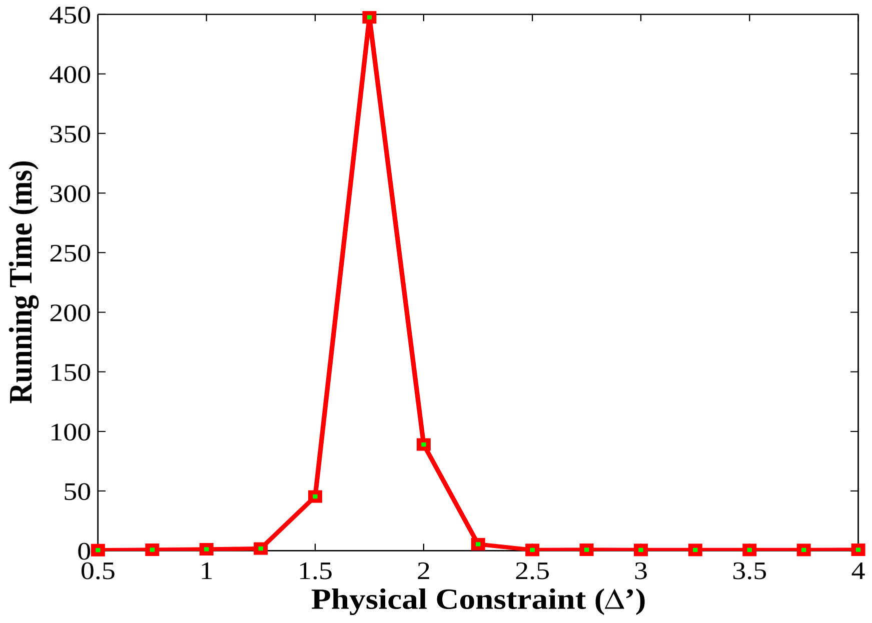


Figure B-9: Running time vs physical constraint for EIOQRA on the random networks with $N = 49$ and $N_R = 8$.

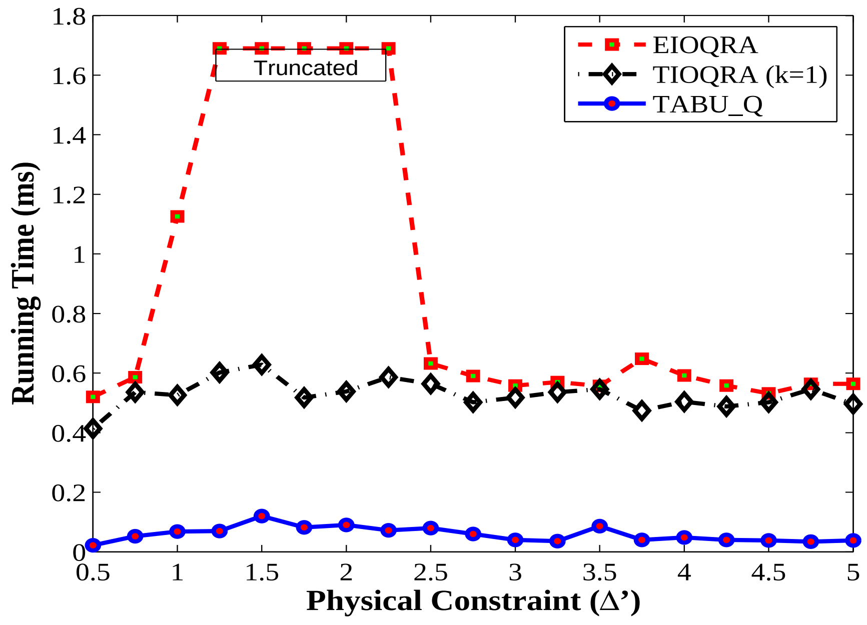


Figure B-10: Running time vs physical constraint for the random networks with $N = 49$ and $N_R = 8$.

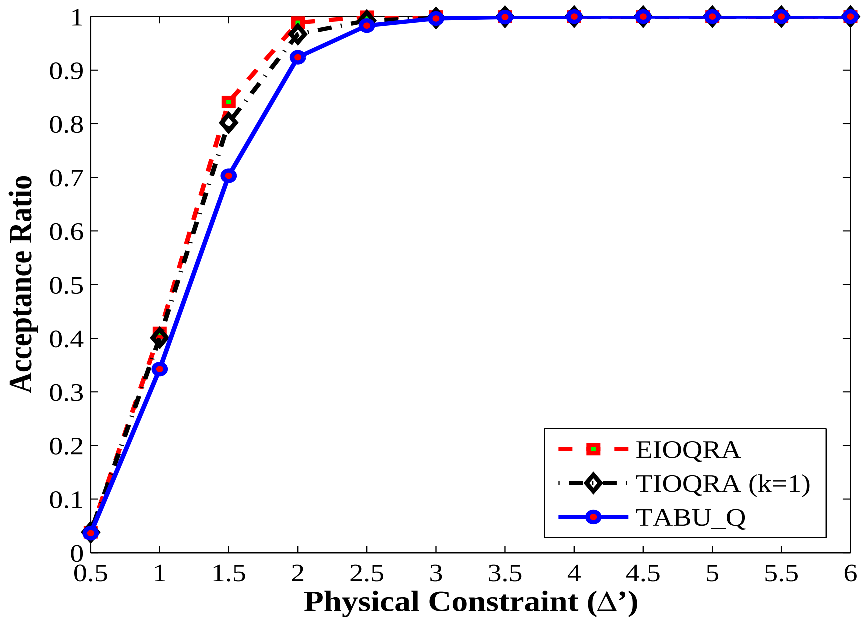


Figure B-11: Acceptance ratio vs physical constraint for the ARPANET with $N_R = 8$.

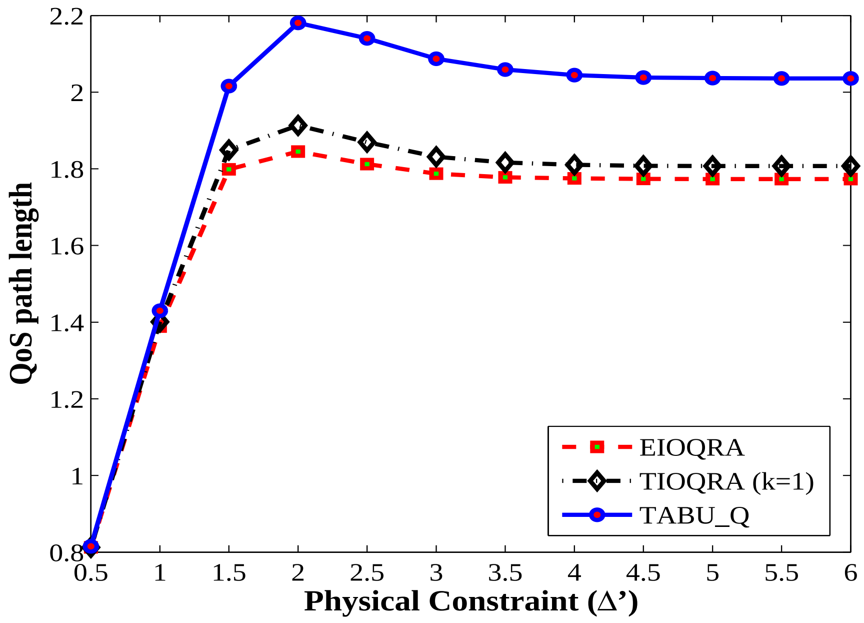


Figure B-12: QoS path length vs physical constraint for the ARPANET with $N_R = 8$.

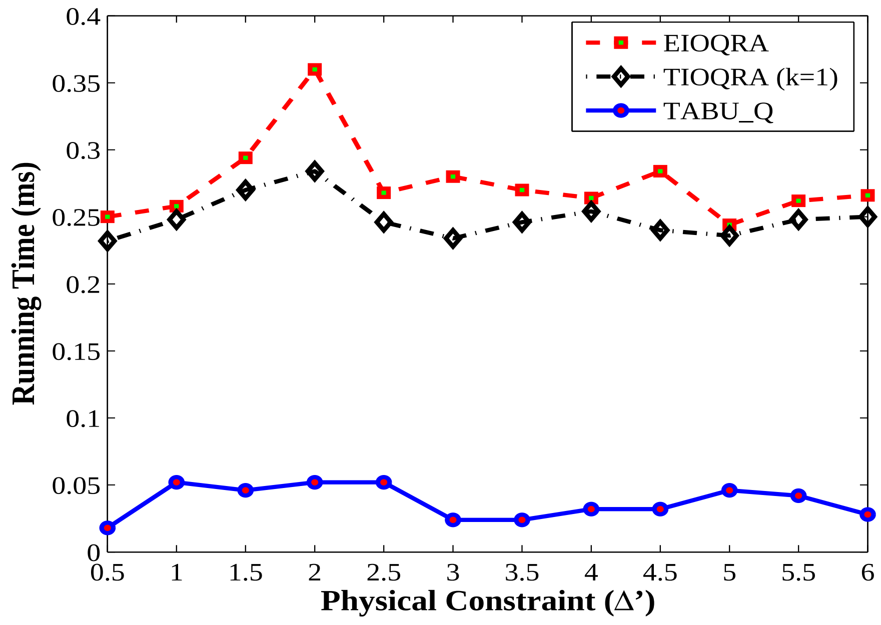


Figure B-13: Running time vs physical constraint for the ARPANET with $N_R = 8$ under totally loose QoS constraints.

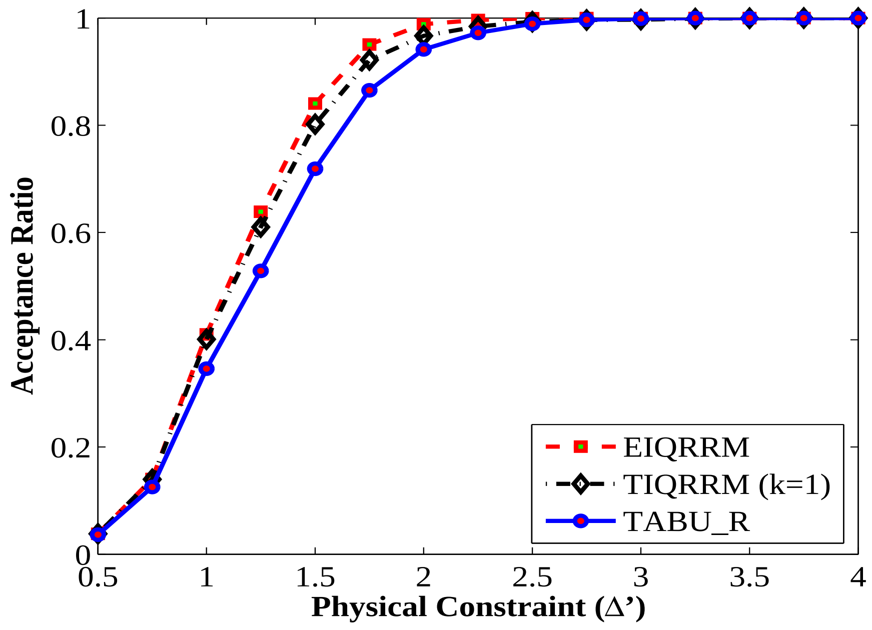


Figure B-14: Acceptance ratio vs physical constraint for the ARPANET with $N_R = 8$ under totally loose QoS constraints.

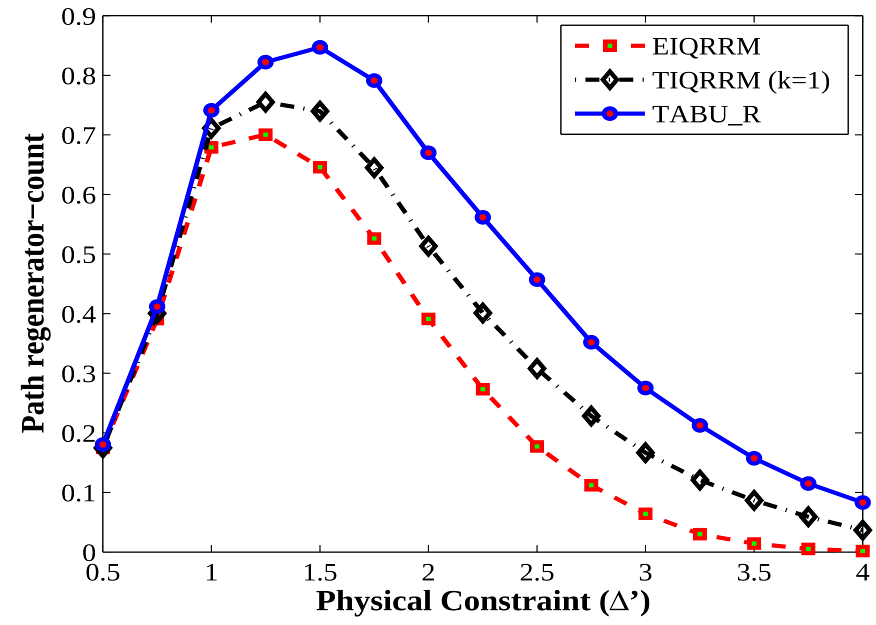


Figure B-15: Path regenerator-count vs physical constraint for the ARPANET with $N_R = 8$ under totally loose QoS constraints.

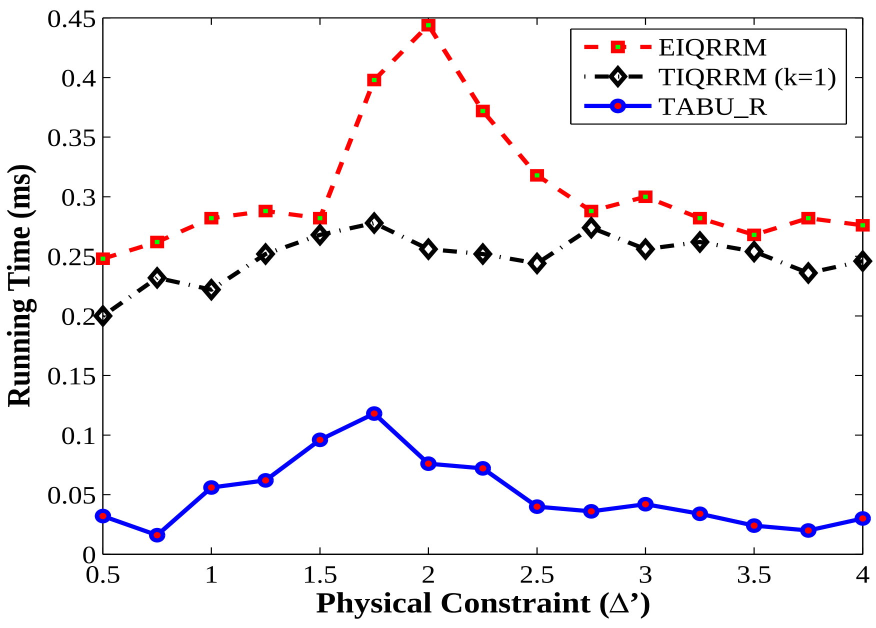


Figure B-16: Running time vs physical constraint for the ARPANET with $N_R = 8$ under totally loose QoS constraints.

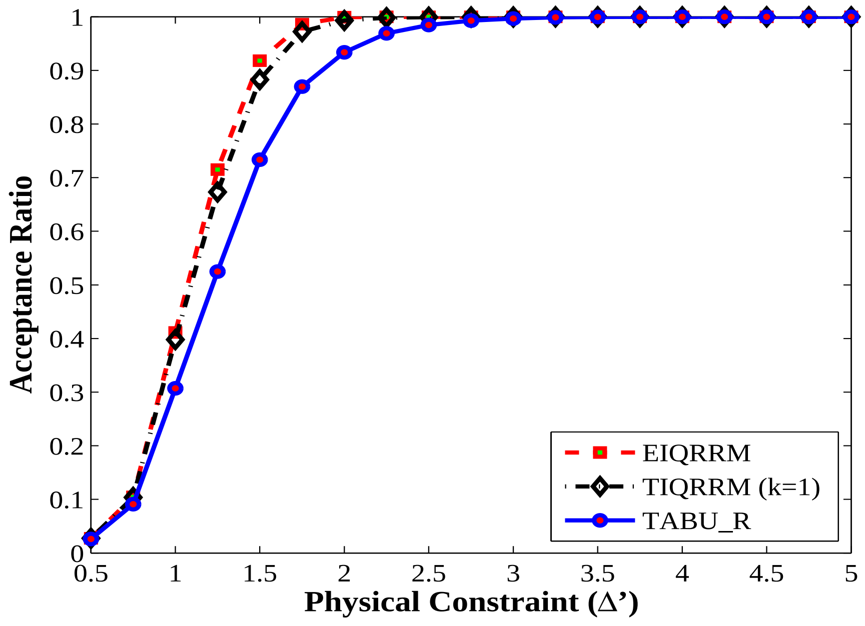


Figure B-17: Acceptance ratio vs physical constraint for the lattice network with $N = 49$ and $N_R = 16$ under totally loose QoS constraints.

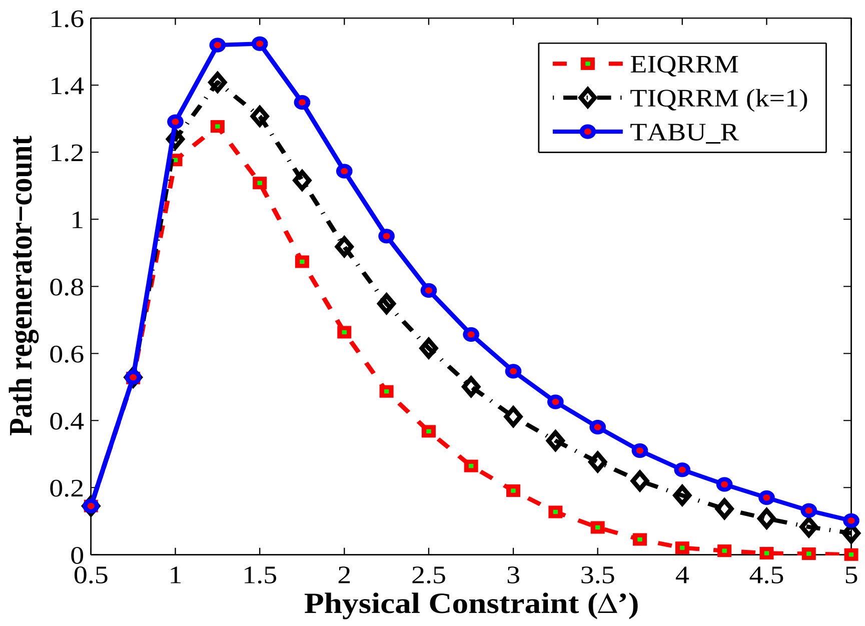


Figure B-18: Path regenerator-count vs physical constraint for the lattice network with $N = 49$ and $N_R = 16$ under totally loose QoS constraints.

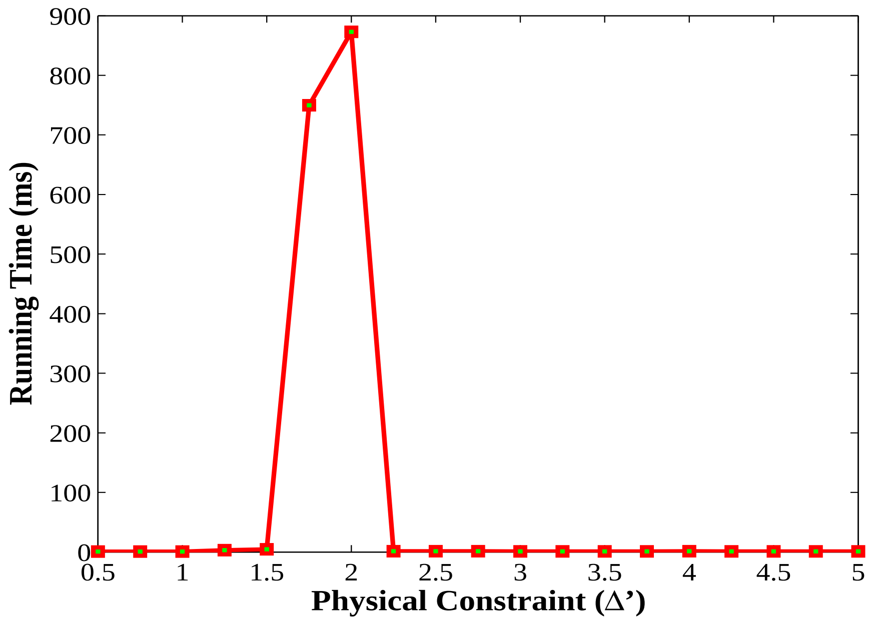


Figure B-19: Running time vs physical constraint of EIQRRM on the lattice network with $N = 49$ and $N_R = 16$ under totally loose QoS constraints.

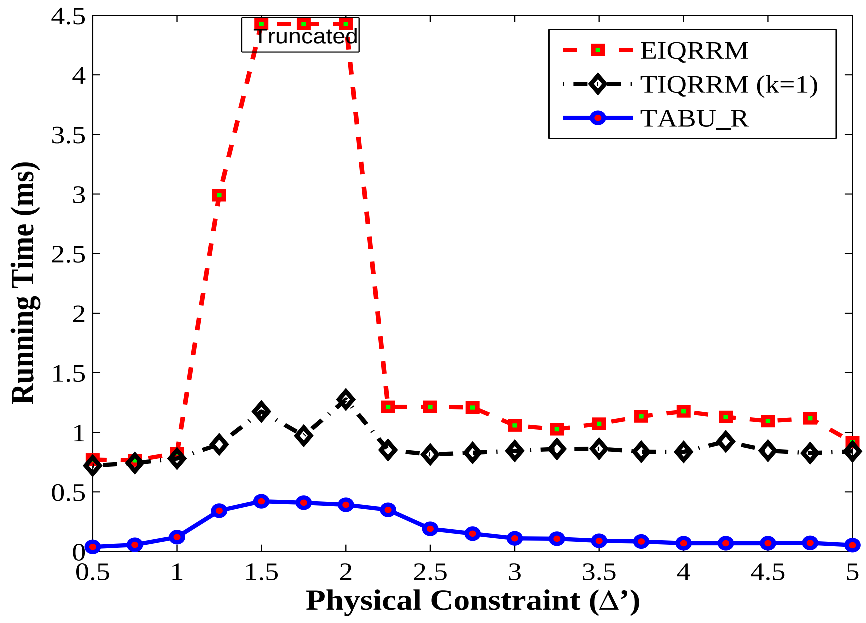


Figure B-20: Running time vs physical constraint for the lattice network with $N = 49$ and $N_R = 16$ under totally loose QoS constraints.

B-3 Tight Qos constraints

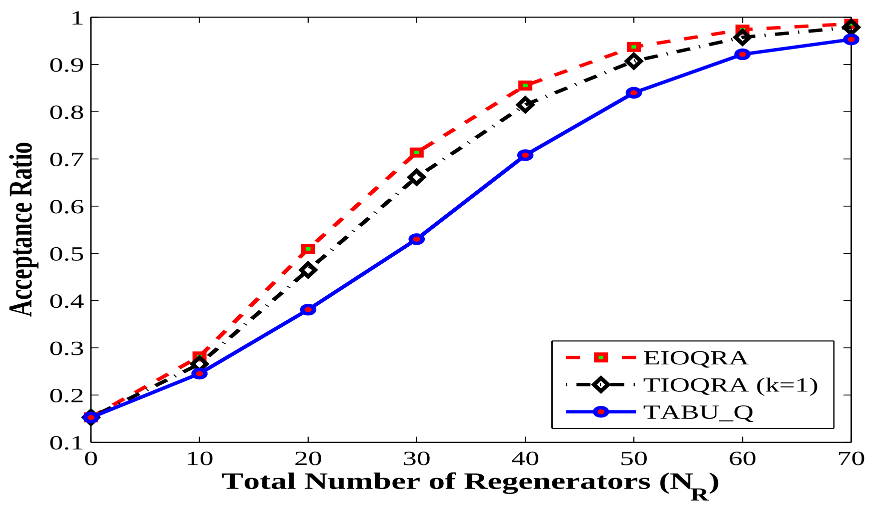


Figure B-21: Acceptance ratio vs total regenerator number for the lattice network with $N = 100$ where $\Delta' = 1.5$ and $T = 6$.

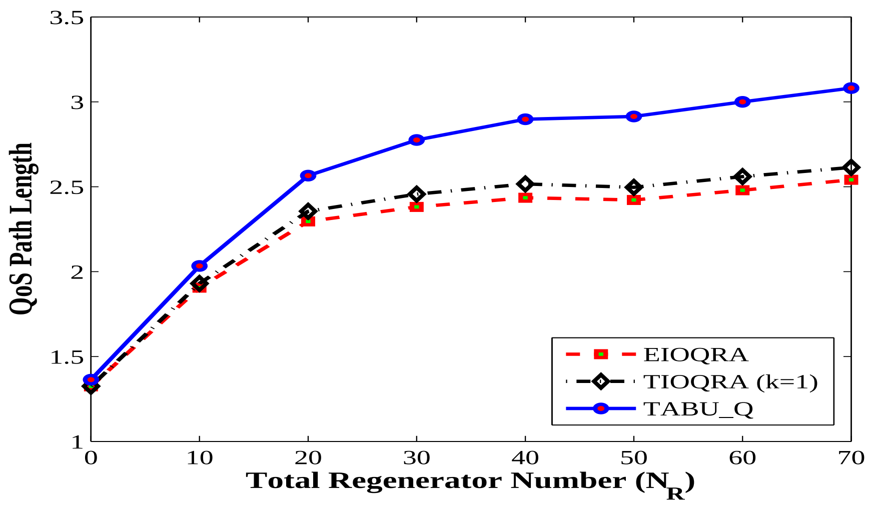


Figure B-22: QoS path length vs total regenerator number for the lattice network with $N = 100$ where $\Delta' = 1.5$ and $T = 6$.

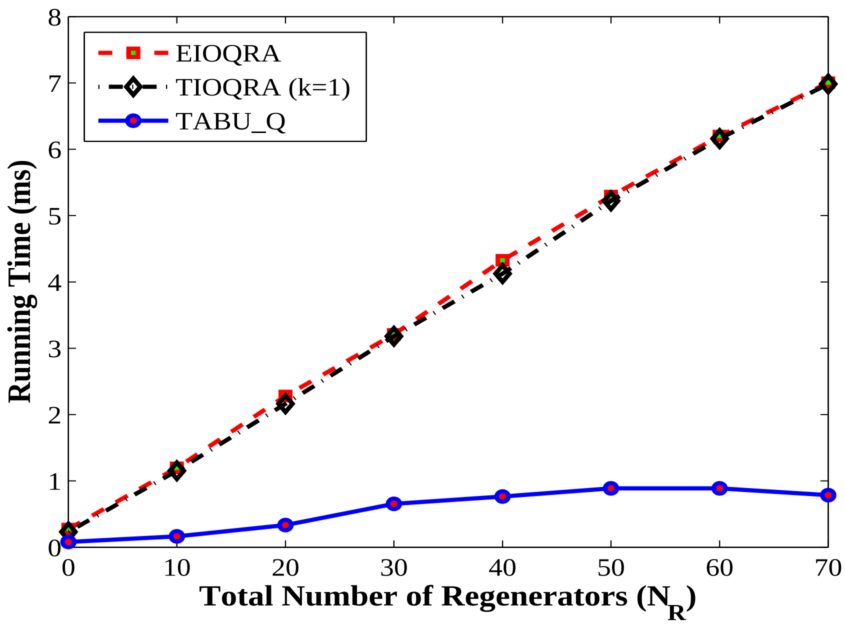


Figure B-23: Running time vs total regenerator number for the lattice network with $N = 100$ where $\Delta' = 1.5$ and $T = 6$.

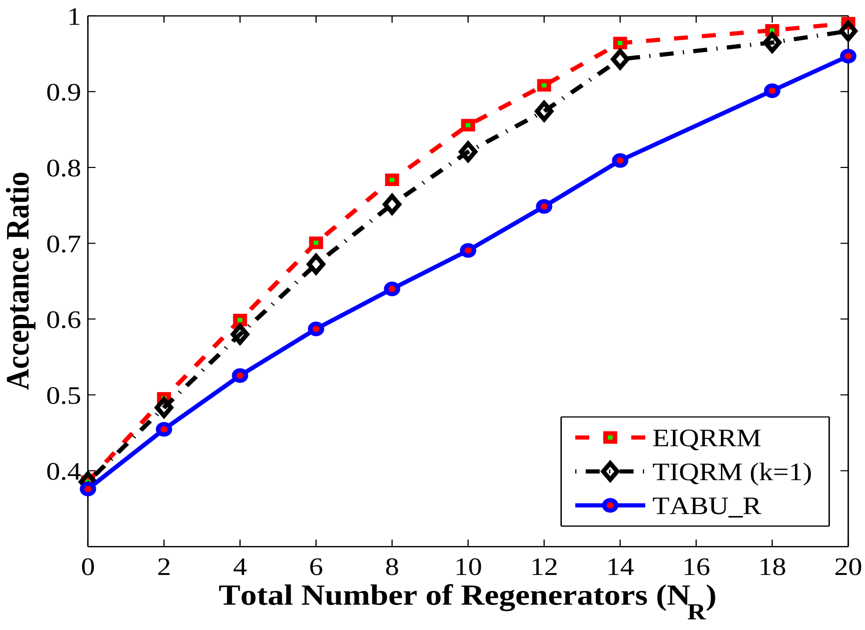


Figure B-24: Acceptance ratio vs total regenerator number for ARPANET where $\Delta' = 1.5$ and $T = 4$.

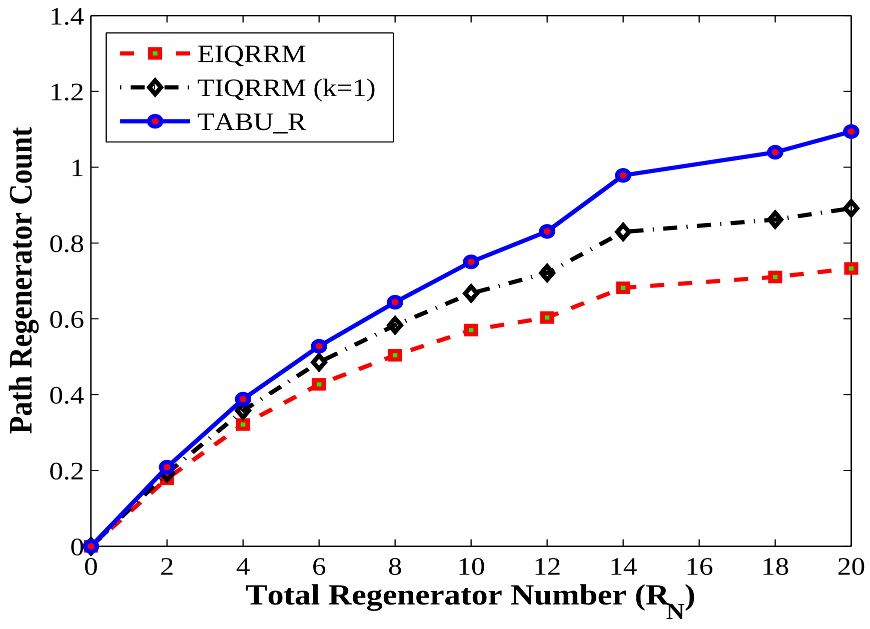


Figure B-25: Path regenerator-count vs total regenerator number for ARPANET where $\Delta' = 1.5$ and $T = 4$.

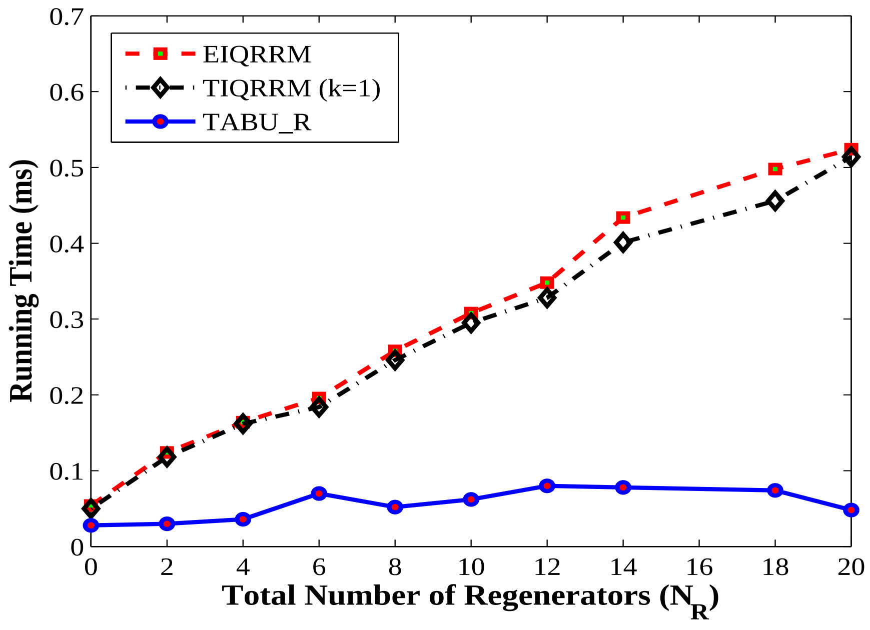


Figure B-26: Running time vs total regenerator number for ARPANET where $\Delta' = 1.5$ and $T = 4$.

B-4 Medium Qos constraints

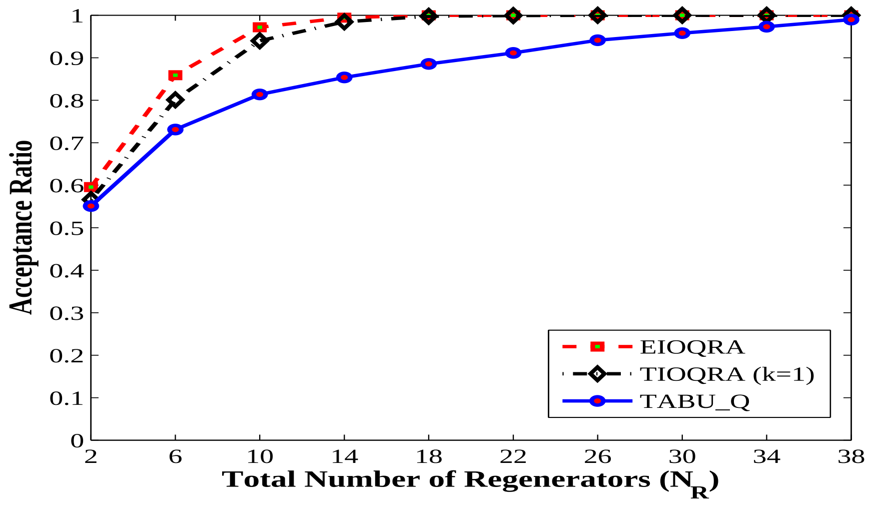


Figure B-27: Acceptance ratio vs total regenerator number for lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

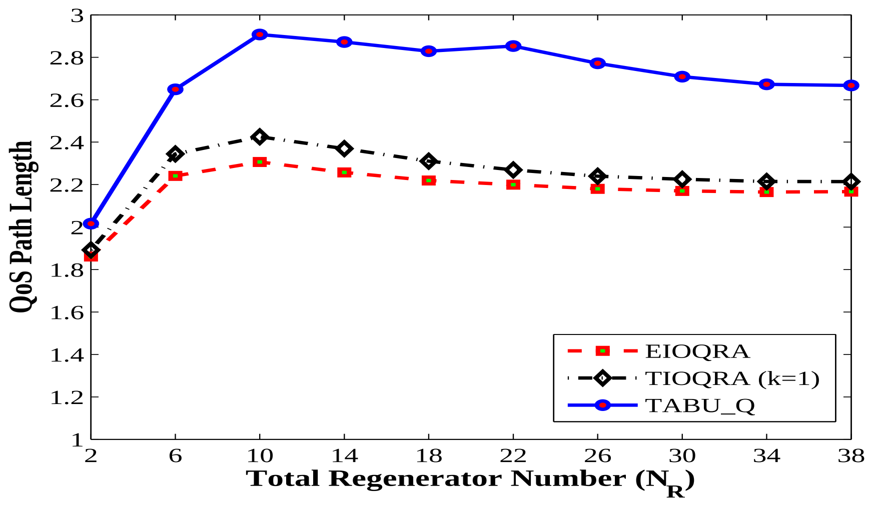


Figure B-28: QoS path length vs total regenerator number for lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

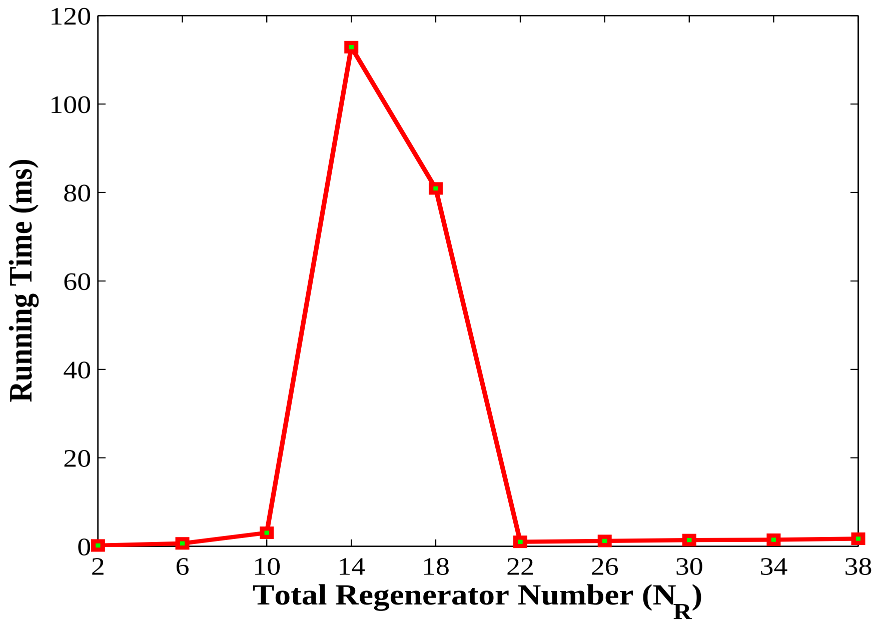


Figure B-29: Running time vs total regenerator number for EIQRRM on lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

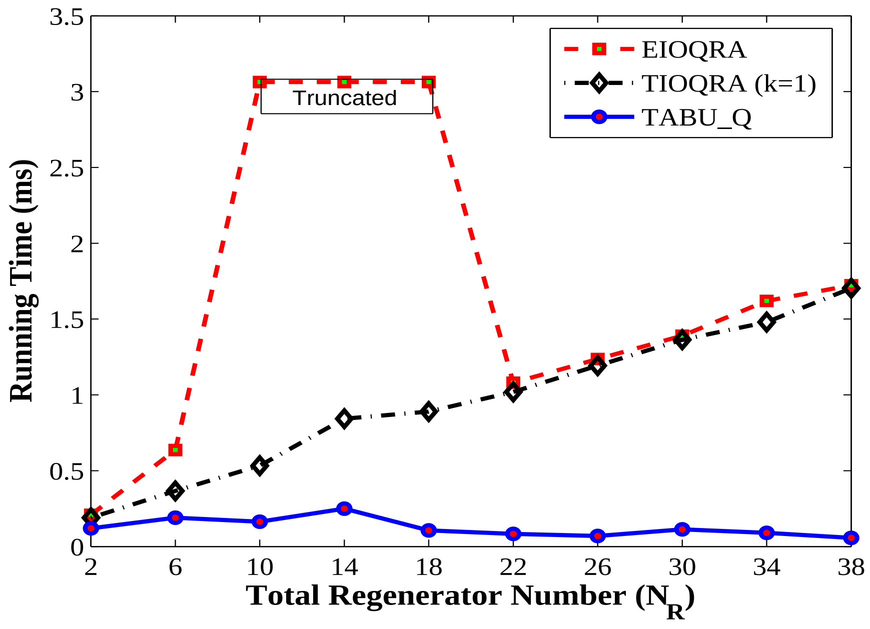


Figure B-30: Running time vs total regenerator number for lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

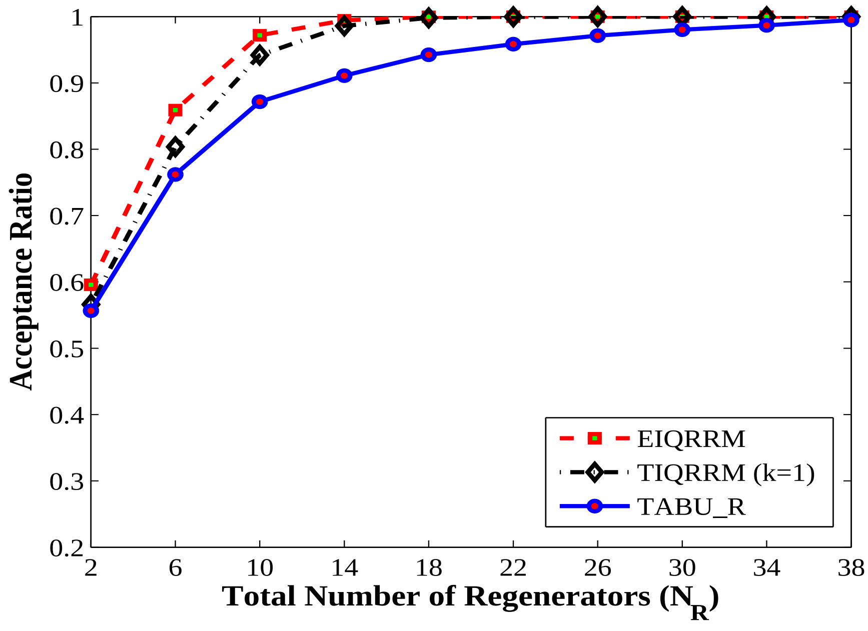


Figure B-31: Acceptance ratio vs total regenerator number for lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

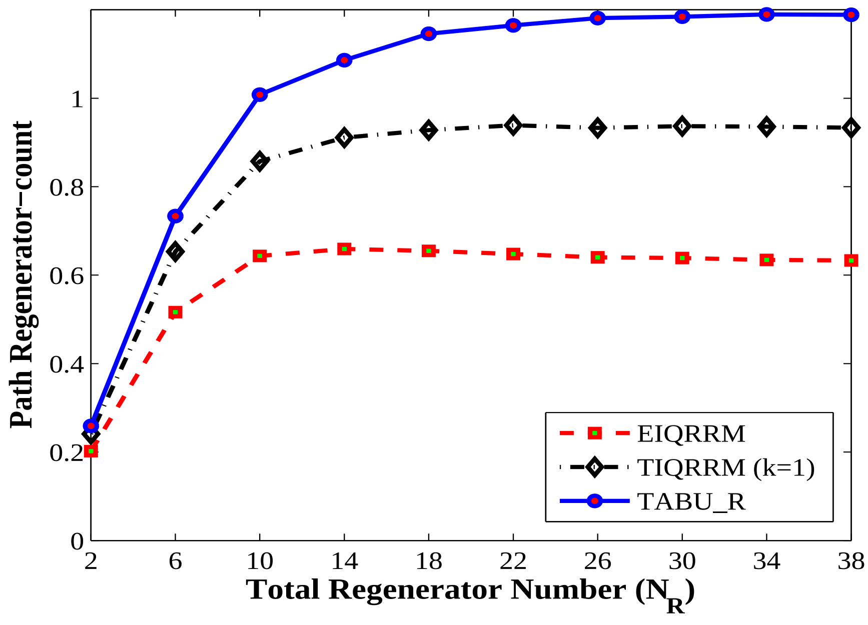


Figure B-32: Path regenerator-count vs total regenerator number for lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

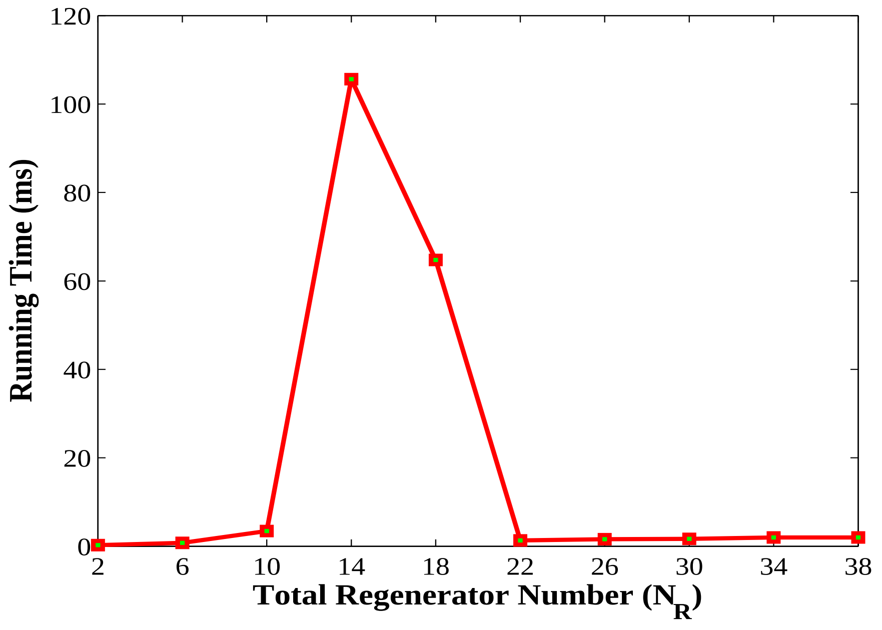


Figure B-33: Running time vs total regenerator number for EIQRRM on lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

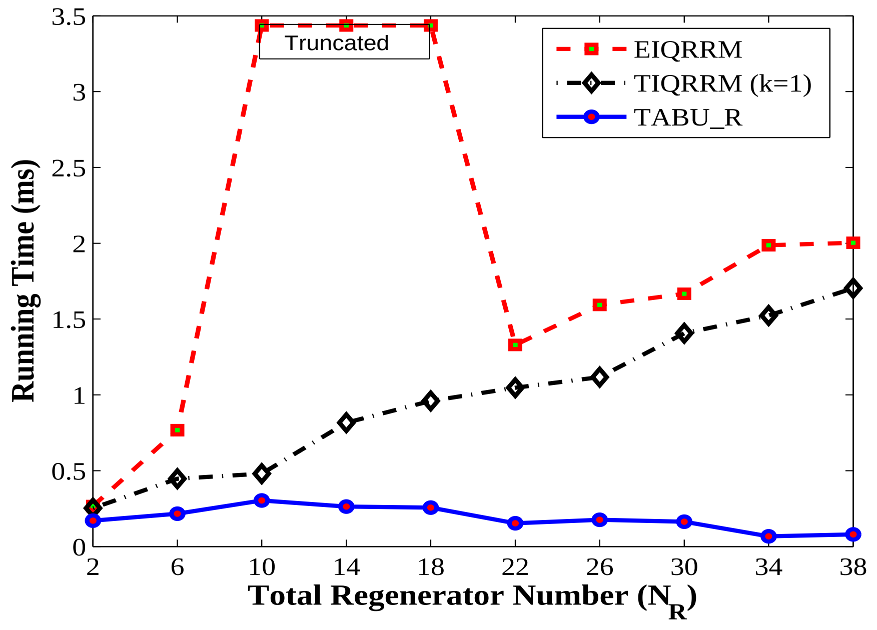


Figure B-34: Running time vs total regenerator number for lattice network of $N = 49$ where $\Delta' = 2$ and $T = 10$.

B-5 Tunable algorithms with different values of k

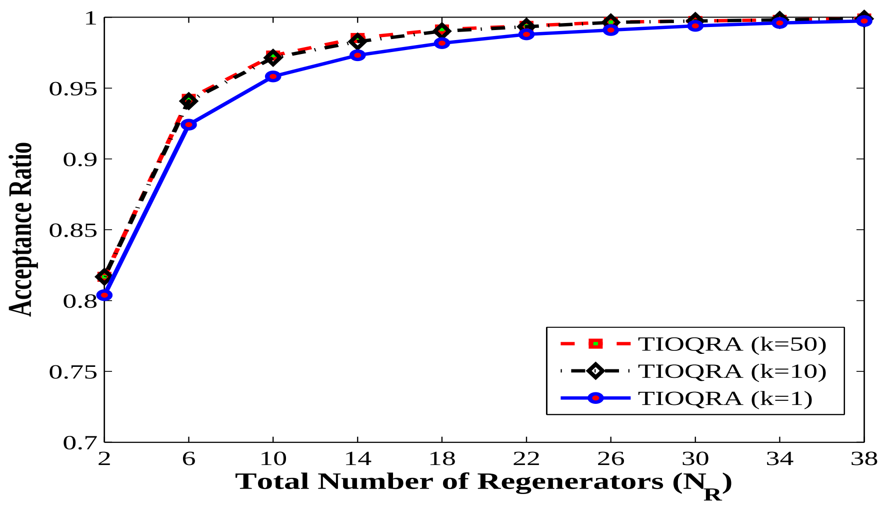


Figure B-35: Acceptance ratio vs total number of regenerators for a random network of $N = 49$ and ρ with $T' = 6$, $\Delta' = 1.5$

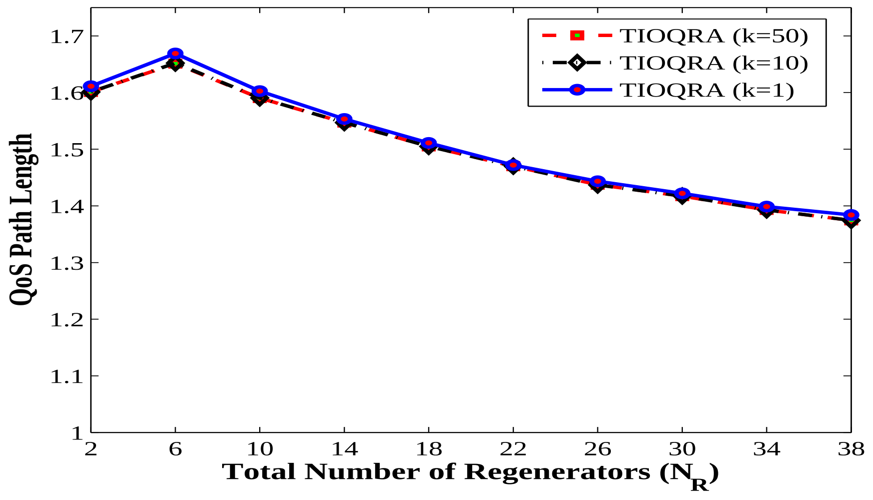


Figure B-36: QoS path length vs total number of regenerators for a random network of $N = 49$ and ρ with $T' = 6$, $\Delta' = 1.5$

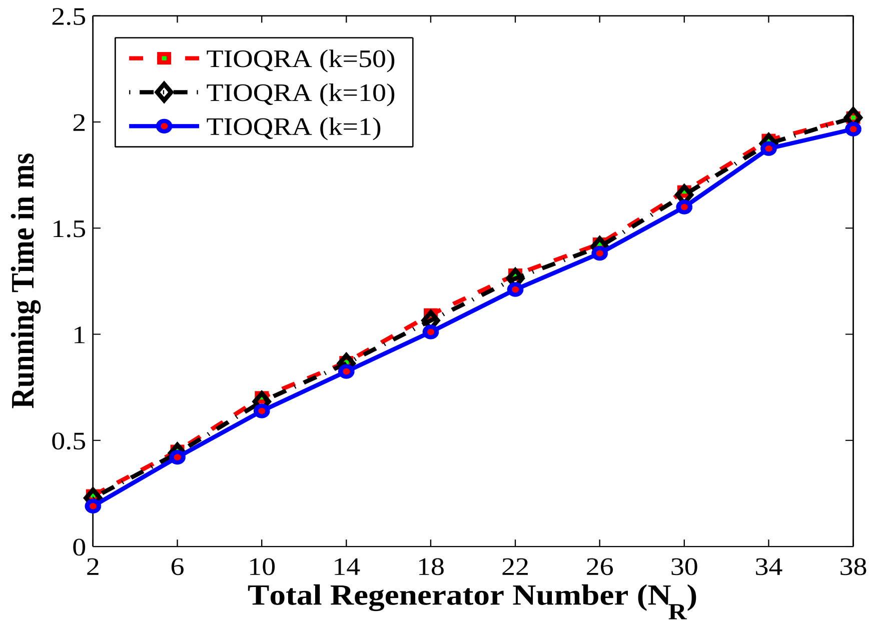


Figure B-37: Running time vs total number of regenerators for a random network of $N = 49$ and ρ with $T' = 6$, $\Delta' = 1.5$

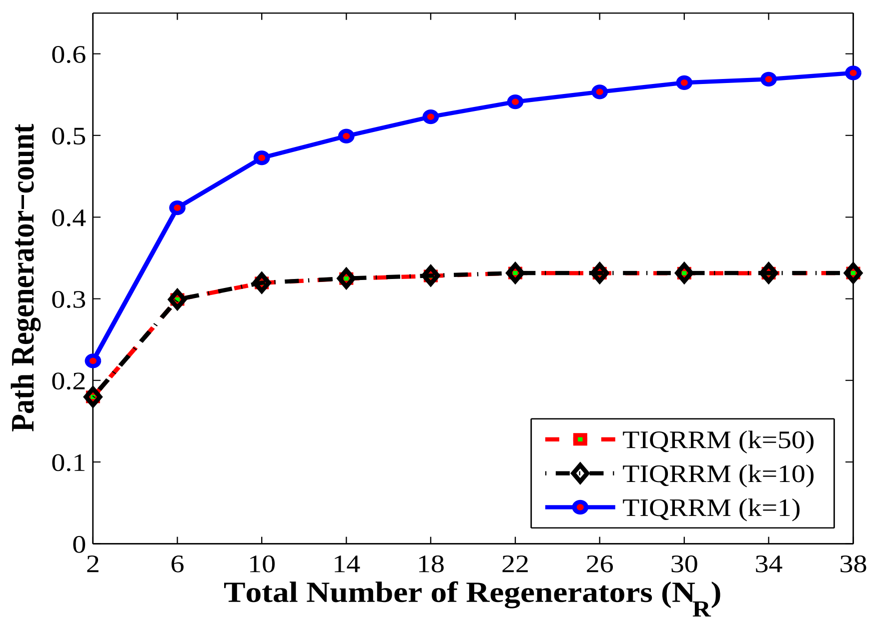


Figure B-38: Path regenerator-count vs total number of regenerators for a random network of $N = 49$ and ρ with $T' = 6$, $\Delta' = 1.5$

B-6 Dynamic traffic

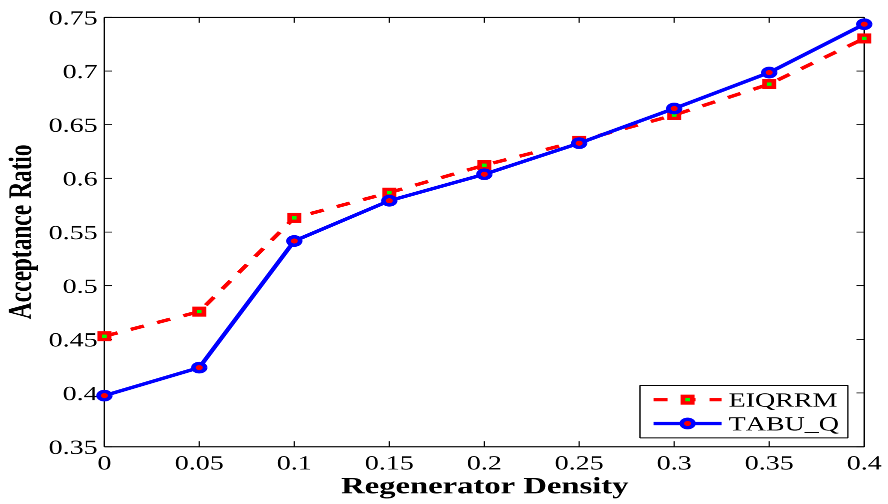


Figure B-39: Acceptance ratio vs regenerator density for a lattice network of $N = 49$ with $W = 20$, $T = 8$, $\Delta' = 1.5$

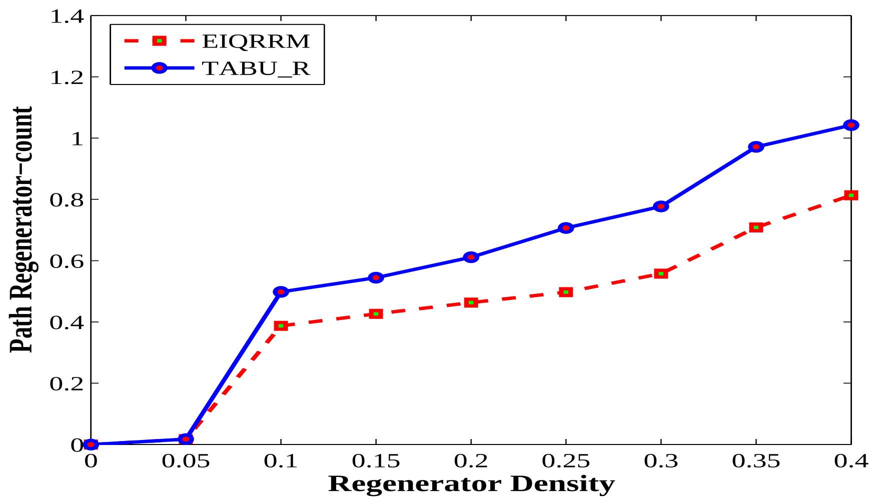


Figure B-40: Path regenerator-count vs regenerator density for a lattice network of $N = 49$ with $W = 20$, $T = 8$, $\Delta' = 1.5$

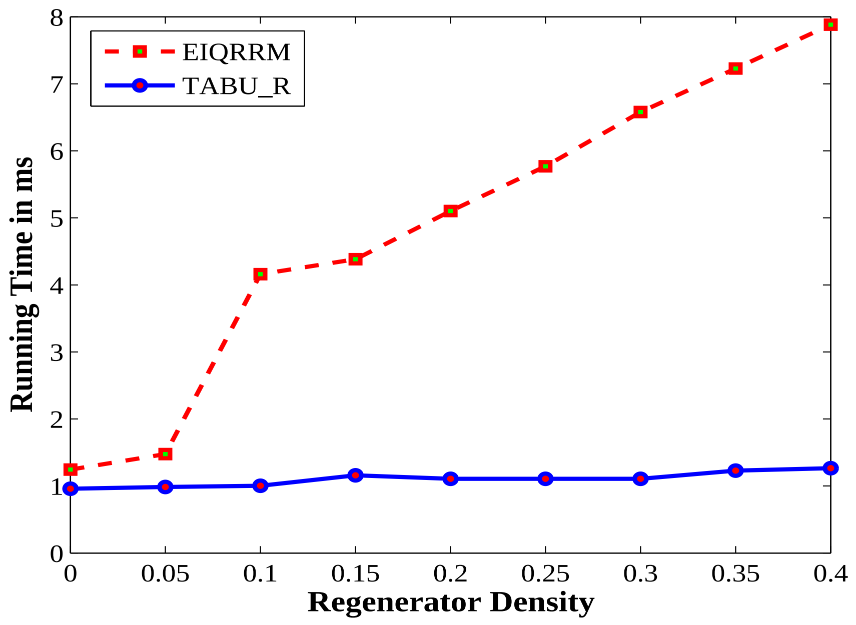


Figure B-41: Running time vs regenerator density for a lattice network of $N = 49$ with $W = 20$, $T = 8$, $\Delta' = 1.5$

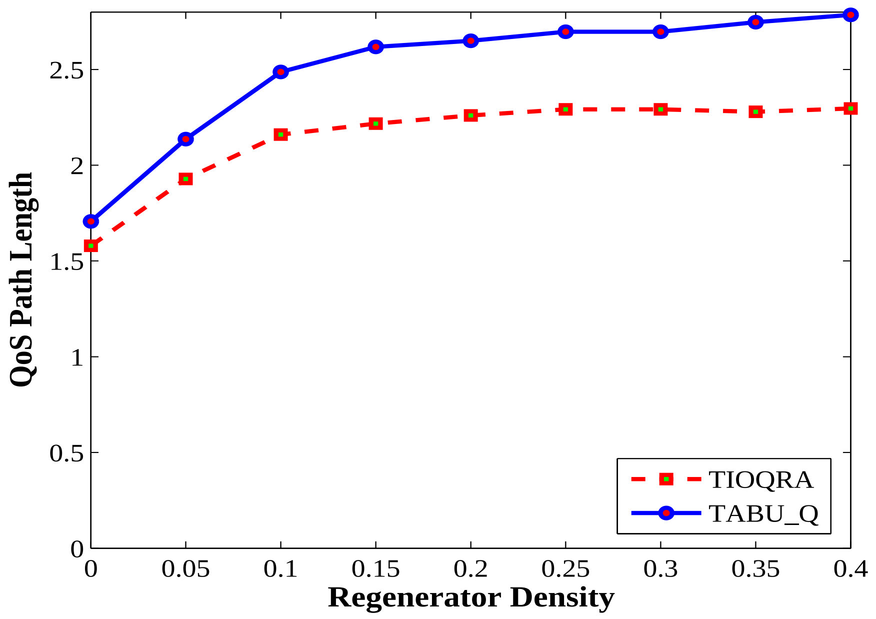


Figure B-42: QoS path length vs regenerator density for a lattice network of $N = 49$ with $W = 20$, $T = 8$, $\Delta' = 1.5$