Photon Netw Commun (2015) 29:307–321
DOI 10.1007/s11107-015-0500-8

Dynamic connection establishment and network re-optimization in flexible optical networks

P. Soumplis[1,2] · **K. Christodoulopoulos**[1,2] · **E. Varvarigos**[1,2]

Received: 4 July 2014 / Accepted: 23 March 2015 / Published online: 8 April 2015

Abstract We consider the problem of dynamic connection establishment and spectrum defragmentation in flexible optical networks. When the spectrum is fragmented, blocking a connection establishment, the algorithm reactively re-optimizes the network by shifting ("pushing") in the spectrum domain and/or rerouting existing connections. We start by presenting an algorithm based on integer linear programming formulation that searches among all combinations of shiftings and reroutings and selects the one that minimizes the changes in existing connections. We also present a heuristic algorithm that recursively shifts/reroutes connections around a void. The solution space of the heuristic can also be very large, so we use a threshold on the recursion depth to reduce the complexity and also provide a trade-off between performance and running time. Our simulation results show that the blocking probability can be substantially reduced using the proposed techniques as opposed to a network that does not reactively defragments the spectrum. The proposed heuristic achieves near-optimal performance, for cases that we were able to find optimal solutions, while the selection of the recursion threshold was shown to provide a good trade-off of performance for running time.

✉ P. Soumplis
soumplis@ceid.upatras.gr

K. Christodoulopoulos
kchristodou@ceid.upatras.gr

E. Varvarigos
manos@ceid.upatras.gr

[1] Department of Computer Engineering and Informatics, University of Patras, Patra, Greece

[2] Computer Technology Institute and Press – Diophantus, Patra, Greece

Keywords Flexible/elastic optical networks · Dynamic Routing and Spectrum Allocation · Spectrum defragmentation · Recursive process

1 Introduction

Flexible optical networks (the term elastic is also widely used) are regarded as the most promising architecture for next generation backbone and metro-area networks, since their increased spectral efficiency and adaptability is considered suitable for future requirements. These networks are based on the flex-grid technology where the spectrum is divided into 12.5 GHz spectrum slots, a smaller granularity than in traditional WDM networks. Moreover, the slots can be combined to create channels that are as wide as needed. With the bandwidth variable transponders (BVT) that can adapt their transmission parameters, flexible networks become more dynamic, adaptive and efficient than traditional solutions [1].

During the operation of an optical network, new connections are established and torn down dynamically with time. In contrast to traditional WDM networks, where spectrum assignment is uniform in the form of wavelengths, in flexible networks the spectrum eventually becomes fragmented, a problem that becomes more severe as time progresses. By spectrum fragmentation, we mean the discontinuity of the available spectrum in the sense that a new connection may not be able to find a desired (same) set of spectrum slots on all the links of its path even though adequate unused spectrum is available (but not at the same central frequency) on the links. Thus, after a point the available spectrum is inefficiently utilized, the network serves fewer demands than one would expect at its actual load level, and connections are blocked even though there is enough spectrum on the links that could be used to serve them.

To address this problem, two types of defragmentation methods have appeared: proactive and reactive methods. Reactive defragmentation is triggered when a new demand cannot be served, while proactive defragmentation is performed in a periodic or in an event-driven manner without being triggered by connection blocking (e.g. at connection release). The latter method aims at maintaining the network in a good shape without a priori knowing whether the changes made will be needed or not. The former method is triggered only when needed, meaning that spectrum fragmentation has reached a critical point, and thus it has to be fast and efficient. Rearranging as few connections as possible and achieving low running time and little disruption in the network are the objectives that matter most in this case.

In this paper, we propose dynamic RSA (D-RSA) algorithms for establishing transparent (without regenerators) and translucent (with regenerators) connections in a flexible optical network that reactively defragment the network when deemed appropriate. We assume an optical network that encompasses slotted flex-grid and tunable transponders, and a generic traffic scenario where demands arrive dynamically, each requesting a specific rate between a specific source and destination. Serving this demand requires the establishment of one or several connections, depending on the requested rate, the distance between the end-points and the capabilities of the transponders. So the D-RSA algorithm has to decide on how to break the requested connection demand into connections (if useful), determine where to use regenerators (if needed and if provisioned in the network), and allocate path(s) and spectrum to the connection(s). If the required resources are free, the demand is served and the connection(s) is (are) established; otherwise, we use two defragmentation techniques to reactively re-optimize the network and serve the demand: (1) the push–pull technique, where established connections continue using the same path but are shifted in the spectrum domain without tearing them down, and (2) the rerouting technique, where established connections are torn down and are routed over the same or different paths in a make-before-break manner. The two techniques are also used jointly to achieve even better performance.

The long-term goal of the D-RSA algorithm is to serve the demands so as to minimize network blocking. If a demand cannot be served at the current network configuration state, the algorithm applies the two aforementioned techniques to defragment the spectrum and re-optimize the network. A secondary objective during the re-optimization is to affect as little as possible the current state of the network, that is, to minimize the disruptions/changes in existing connections. We define the re-optimization cost in terms of the number of connections that are shifted by the push–pull technique or in terms of the number of rerouted connections for the rerouting technique. We devised an ILP algorithm that serves a demand searching every possible combination of reroutings and shiftings of the established connections and finally selecting the one with the minimum cost. Since the problem is NP-hard and the solution space is huge, we also developed an appropriate heuristic algorithm. The heuristic selects a spectrum void that if expanded could serve the new connection and then searches among different shifting and rerouting combinations of the void's adjacent connections, to finally select the combination with the minimum cost. Since shifting or rerouting a connection might in turn trigger more shifting and rerouting actions, the proposed heuristic is recursive, and the solution space can be also very large. To control the running time, we use a threshold on the recursion depth. Our simulation results show that the blocking probability can be substantially reduced using the proposed techniques. The proposed heuristic achieves near-optimal performance, comparable to that obtained by the ILP algorithm, at least for the small size network experiments for which we were able to track optimal ILP solutions. Finally, the selection of the recursion threshold was shown to provide a good trade-off of performance for running time.

The rest of the paper is organized as follows. In Sect. 2, we report on the related work. In Sect. 3, we formally define the dynamic connection establishment and defragmentation problem under study. In Sect. 4, we present our solutions and in particular the ILP formulations and also our heuristic algorithms. In Sect. 5, we present the performance comparison results. Our conclusions follow in Sect. 6.

2 Related work

Dynamic flexible optical networks have received increased recent attention, with much of the research effort focusing on algorithms to cope with connection establishment and spectrum fragmentation [2]. One way to reduce spectrum fragmentation is to re-optimize the network by rerouting (tearing down and re-establishing) existing connections. A second and often better approach is to use the finer granularity and channel adaptability offered by the advanced transponders envisioned in order to shift connections in the spectrum domain without interruption. This is demonstrated in [3] by the so called *push–pull* technique.

Irrespectively of the method they use (rerouting or spectrum shifting), the corresponding defragmentation algorithms can be divided, as mentioned before, into reactive [8–11] and proactive [4–7] algorithms, according to whether they are triggered or not by a blocking/critical event.

The authors in [8] defragment the spectrum by rerouting existing connections so as to pack them into the lower spectrum slots, while also minimizing connection interruptions. In [9], different techniques for spectrum sharing between neighbouring connections are introduced to serve time varying traffic. The authors in [10] follow the blocked-triggered

approach and provoke the defragmentation algorithm each time there are not enough resources for a new connection. The difference to our approach is that the algorithm proposed in [10] minimizes the connections that are affected without distinguishing between spectrum shifting (push–pull) and rerouting and without taking into consideration the special characteristics (time and cost) to make use of these operations. What is however missing in our case is a full network system experiment, as performed in [11], where the authors extended their work in [10] and integrated the algorithm in an application-based network operations (ABNO) network controller to deal with the spectrum shifting defragmentation. In [5], the authors propose an algorithm that is invoked when a connection or a group of connections are torn down, by rerouting-associated remaining connections at lower spectrum bands. In [6], the authors examine defragmentation in practice by rearranging connections spectrally while also considering the advantages obtained by different channel spacing selections. In [7], two defragmentation methods are proposed, with the first focusing on the most congested links, and the second performing network-wide proactive defragmentation by rerouting connections so as to pack them in a most-used spectrum slot assignment manner. A reactive defragmentation method is proposed in [8], where blocking triggers the rerouting of existing connections to make space for the new connection.

The novelty of our proposed solutions compared to previous works is fivefold. First, we provide general algorithms that take generic parameters as input. In particular, the input comes in the form of feasible transmission configurations of the transponders used in the network, which incorporate and account for the physical layer impairments. Second, previous work focused on transparent networks, without, to the best of our knowledge, considering regenerators. Our algorithms take into account regenerators that can be used to achieve higher spectrum efficiency and reduce blocking probability. Third, previous works perform defragmentation using either push–pull or rerouting, while in our work we also consider the combination of these techniques to achieve even better performance. We explore a wider defragmentation space than former approaches, by examining all possible combinations in the proposed ILP algorithm, finding the optimal reactive defragmentation solution if one exists. Finally, we also propose a heuristic that uses a threshold on the recursion depth to control the complexity and trade off performance for running time.

3 Problem description

We are given an optical network $G = (V, E)$, where V denotes the set of nodes and E denotes the set of single-fibre links. Each link $l \in E$ is characterized by its length D_l in km. The spectrum is divided into spectrum slots of F GHz, where one spectrum slot corresponds to the switching granularity of the flexible network elements (flex-grid switches and bandwidth variable transponders—BVTs). The network supports a total of F_t slots on a link.

The traffic is served by tunable BVT transponders that control (a) the modulation format and (b) the spectrum (in the form of contiguous spectrum slots) they utilize. By adapting these features, a BVT of cost c can be tuned to transmit at a rate of r Gbps using bandwidth of b spectrum slots and a guardband of g spectrum slots from the adjacent spectrum connections to reach a distance of l km with acceptable quality of transmission (QoT). More formally, a specific transponder of cost (type) c is characterized by its physical feasibility function f_c that gives the reach $l = f_c(r, b, g)$ at which it can transmit with acceptable QoT as a function of the parameters r (rate), b (spectrum) and g (guardband) that we can control. This function captures the physical layer impairments, assuming the worst case contribution for the interference-related impairments (four-wave-mixing, cross-phase modulation, crosstalk), and can be obtained either through experiments or using analytical models [8,13].

Using function f_c, we define (*reach-rate–spectrum–guardband–cost*) transmission *tuples* $t = (l_t, r_t, b_t, g_{t,}, c_t)$ that correspond to *feasible* transmission configurations of the specific transponder. The term "feasible" is used to signify that the tuple definition incorporates the limitations posed by physical layer impairments. The transponders have certain limitations in their capabilities, which are of the following forms: the maximum symbols per second (baud rate), the maximum modulation format, the maximum spectrum used, and/or the maximum transmission rate. Given the transponders' limitations, and since the modulation format and the spectrum are selected from discrete sets, we obtain the set of feasible transmission configurations for the available transponders.

We assume that demands for new connections arrive at random time instants and are immediately served by the dynamic routing and spectrum allocation (D-RSA) algorithm. A demand θ is characterized by its source-destination pair (s_θ, d_θ), its requested capacity Λ_θ and a mode indicator M_θ that states if the demand is allowed or not to use regenerators (translucent or transparent, respectively). This definition is quite generic and can capture traffic serving in both transparent and translucent networks, but also allows for extra functionality as will be discussed shortly. If the demand cannot be satisfied in the current state of the network, the network is re-optimized. Thus, the problem can be viewed as an extension of the offline RSA and thus is NP-hard. In this paper, we propose an exact ILP formulation and a heuristic D-RSA algorithm to provide solutions for large problem instances.

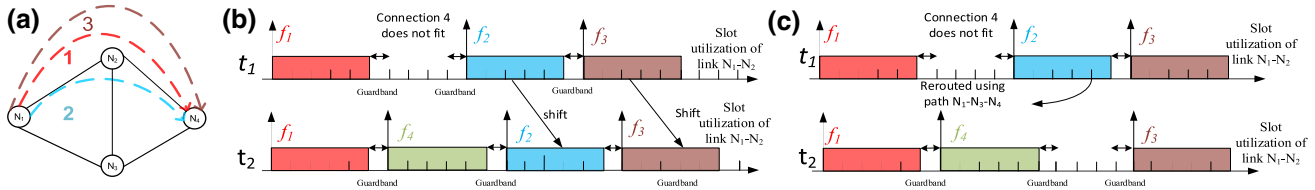

Fig. 1 **a** A flex-grid network where three connections exist at time t1. **b** The spectrum slot allocation on link N1–N2 at two different time instants t1 and t2. Initially at time t1, connections 1, 2 and 3 exist and a new connection 4 arrives. Not finding sufficient space to serve it, we shift (push–pull) connections 2 and 3 by two slots each, and the new connection 4 is established at time t2. **c** Same scenario but using the rerouting technique

Connection establishment in a network where the use of regenerators is not provisioned can be performed by requiring all demands to be served transparently. In a translucent network, if the mode indicator of a demand allows the use of regenerators, it is up to the D-RSA algorithm to decide whether to use them or not. However, the extra functionality comes when a demand in a translucent network explicitly requires transparent service (e.g. for reduced cost), and this is performed by the proposed D-RSA by providing an appropriate non-regenerated solution.

To serve a demand θ between s_θ and d_θ requesting rate Λ_θ, the D-RSA establishes one or more parallel (in case rate Λ_θ is not supported at the respective distance by a single transponder) transparent or translucent connections from s_θ to d_θ. Note that translucent connections consist of transparent subconnections (sub-paths) regenerated at intermediate points, and thus the transparent case can be viewed as a special case of the translucent case. In the remainder of this paper, we will avoid specifying the mode (transparent or translucent) of a connection unless this is not apparent from the context.

The D-RSA algorithm examines a number of candidate paths between the source and destination, and depending on lengths of the links that comprise them, it finds the transmission tuples that can be used over each of them, what we call the set of feasible path-tuple pairs. For a given demand, some path-tuples will never be used (said to be "dominated" by others), since they require more spectrum and transponders than some other path-tuple. These are removed, limiting the search space without losing good solutions.

For each non-dominated path-tuple pair, since the requested rate Λ_θ can be higher than the maximum transmission rate at the corresponding distance, the algorithm calculates the number of (sub-)connections to be established, the nodes where regenerators will be placed, if needed and allowed, and the amount of spectrum to be used by each connection. Then it searches to allocate contiguous spectrum slots to these connections. Note that each path-tuple corresponds to a specific number of connections and regenerators, and by examining all the path-tuple pairs we examine all feasible combinations of these parameters. For example, we examine whether a single high-rate short-reach connection requiring many regenerators at intermediate nodes should be used or whether the other extreme that uses many low-rate long-reach connections should be used, or whether some other option in-between is preferable.

If the algorithm finds sufficient available spectrum slots for a specific path-tuple pair, the connection(s) defined by the path-tuple pair choice is (are) established, subject to the RSA limitations. If the number of free spectrum slots is insufficient for at least one connection, the (1) push–pull or (2) rerouting, or a combination of these techniques is used to reactively re-optimize the network, free spectrum and serve that connection. The push–pull technique offers the ability to shift an existing connection's spectrum band with no service disruption (hitless shifting) and can be used to defragment the network [3]. Figure 1 presents an example where the push–pull technique is used to establish a connection that would otherwise be blocked. The time to push–pull an existing connection is proportional to the number of slots by which the connection is shifted, and experiments show that this can be done quite fast. We assume that shifting more than one connection that is adjacent towards the same direction can be done in parallel, but in general, the algorithm's goal is to avoid disruption, affecting as few connections as possible.

The rerouting technique we assume is based on the make-before-break (MbB) technique and utilizes additional transponders (and regenerators, when used) to re-establish an existing connection before tearing it down. Thus, before rerouting an existing connection, we re-establish it over the same or a different path. In some cases, we may have to utilize more than one transponders to make feasible the transmission over the new path. The old and the new connections will both be active for a small interval of time, while the traffic is switched from the old to the new one(s). Finally, the old connection is torn down releasing the spectrum that had been reserved for it. Similar to the push–pull technique, when more than one connections are rerouted we assume that this is done in a pipelined manner, minimizing the number of spare transponders/regenerators required. The MbB technique guarantees small traffic disruption (this depends on the difference in the lengths of the two paths used and would be seamless if we reroute the connection over the same path). Compared to the push–pull, when rerouting a connection we may end up establishing more than one connections

and use more regenerators than before. The downside of the push–pull technique is the inability to change the path of an already established connection, which may be necessary in some cases, and is achieved using the rerouting technique. In other words, the push–pull technique allows changes only in the spectrum and not in the space domain when trying to reduce defragmentation.

The ILP algorithm we devised searches all possible shifting and rerouting combinations of the active connections, thus searching both spectrum and space reconfiguration options. By minimizing the corresponding cost in terms of a weighted combination of the number of rerouting and shifting actions performed, it keeps the network state as close as possible to the previous state, while finds sufficient space to serve the new demand. The search space for the ILP algorithm can be enormous, depending on the number of active connections. This problem is dealt effectively by the heuristic algorithm that explores a much smaller search space.

The heuristic algorithm we devised selects the biggest void and makes space around that void, by applying the shifting and/or rerouting techniques, in a recursive manner. The shifting can be done towards the upper and bottom direction by certain slots, until we create the required spectrum space. Different selections of the slots that are freed in each direction result in different solutions and costs. To achieve the lowest possible cost, the heuristic algorithm considers all possible combinations of slots freed in the two directions, but only for that particular void. A similar approach is followed when rerouting connections. The heuristic can also combine the two aforementioned techniques and examine cases where some connections are rerouted and others are push–pulled to create the appropriate space, selecting the combination that achieves the lowest cost.

Finally, note that although our algorithms are parametric to the cost of these two techniques, it stands to reason to assume that reroutings have higher cost due to the need for spare transponders and the disruption of service (depending on path lengths) that might be experienced in certain cases.

4 D-RSA algorithms description

In this section, we describe the proposed algorithms for the dynamic routing and spectrum allocation (D-RSA) problem. The D-RSA algorithm is executed each time a new demand arrives. If serving this demand is not possible under the current network state, the algorithm reactively defragments the network with the objective of making the fewest changes to the active connections. We present in Sect. 4.1 an ILP formulation and then in Sect. 4.2 a heuristic D-RSA algorithm to perform this task.

Both the ILP and heuristic D-RSA algorithms are invoked when a new demand θ described by $(s_\theta,\ d_\theta,\ \Lambda_\theta, M_\theta)$ arrives, where $(s_\theta,\ d_\theta)$ are the source and destination, Λ_θ is the demanded capacity in Gbps, and M_θ is the mode indicator that specifies whether the demand has to be served transparently or not. In order to reduce the time required to serve a demand, for every (s, d) pair in the network we pre-calculate a set P_{sd} of k alternative paths that could be used, using a variation of the k-shortest path algorithm. For each pre-calculated path, we identify the configurations (tuples) that can be used by the transponders over that path, based on the lengths of its constituent links. In particular, we examine whether a feasible transponder configuration tuple $t = (l_t,\ r_t, b_t, g_{t,}, c_t)$ has transmission reach l_t higher than the length of the path p for the transparent case or higher than the maximum link length of path p for the translucent case. Note that a translucent (sub-)connection is terminated at a regenerator at an intermediate node and a new (sub-)connection is initiated, to create an end-to-end translucent connection. For the translucent case, for each acceptable path-tuple pair (p, t), the path p is swept from left to right and a regenerator is assumed whenever required, that is, at the last node before the transmission distance l_t of the tuple is reached. Thus, for path-tuple pair (p, t), we find the set of nodes where regenerators have to be placed to make the transmission feasible, splitting the end-to-end translucent path p into sub-paths the set of which is denoted by $R_{p,t}$. In what follows, we will describe the algorithms considering translucent connections and the use of regenerators, since it is more general and contains the only-transparent setting as a subcase.

In the pre-processing phase for each nodes pair (s, d), we have calculated a set Q_{sd} of feasible path-tuple pairs, including the regeneration points. Then, when a demand θ arrives, depending on its requested rate Λ_θ we take each feasible path-tuple pair $(p,\ t)$ from $Q_{s_\theta d_\theta}$ and break the demand into $W_{\theta,p,t}$ parallel connections. $W_{\theta,p,t}$ is equal to one when $\Lambda_\theta \leq r_t$ or higher than one when $\Lambda_\theta > r_t$. That is, we need $W_{\theta,p,t}$ transponders configured at transmission tuple t to serve the demand θ over path p, while the regeneration points $R_{p,t}$ are the same irrespectively of the demanded rate (and thus were pre-calculated). We denote the set of feasible path-tuple pairs for demand θ as Q_θ that include also the regeneration points and number of parallel connection breakings. Each path-transmission pair in Q_θ is a candidate solution to serve the demand. It is the role of the D-RSA algorithm to select the one and allocate spectrum to the related connection(s). Note that a path-tuple pair $(p,\ t) \in Q_\theta$ can include one or more *transparent* connections, denoted by $(p,\ m,\ t,\ i)$, where $m \in R_{p,t}$ and $i \in \{1, 2, \ldots,\ W_{\theta,p,t}\}$, depending on the regeneration points and the number of parallel sub-connections. If the D-RSA algorithm cannot find the spectrum to allocate to all these transparent connections, it reactively defragments the network using the push–pull or the rerouting techniques.

4.1 ILP algorithm

We now present the D-RSA ILP formulation, which is an extension of the ILP formulation presented in [16]. Note that we can go from any previous network state to any new network state with specific shifting and rerouting operations. Thus, the proposed formulation is actually a planning algorithm, and in particular the one presented in [16], with additional constraints to measure the number of shiftings and reroutings of the active connections performed and a different objective function to minimize this (defragmentation-related) cost. In particular, the proposed ILP algorithm takes the new demand together with the previous demands and the connections serving them. We will denote the new demand by $\widehat{\theta}$ and an existing demand by $\overline{\theta}$, while $\overline{\Theta}$ will be the set of existing demand and $\Theta = \overline{\Theta} \cup \{\widehat{\theta}\}$ the set of all demands. The algorithm allocates paths and spectrum to the new demand but also to the previous demands, so it is allowed to change the existing active connections.

To be more specific, the algorithm takes as input the feasible path-tuple pairs for all demands $\theta \in \Theta$ that include the path-tuple pairs and for each the regeneration points (which break the end-to-end path into transparent sub-paths the set of which is denoted by $R_{p,t}$) and the number of breaking of the demands into parallel sub-connections (denoted by $W_{\theta,p,t}$). The active connections in the network are described by their utilized path-tuple pairs $\overline{x}_{\overline{\theta},p,t}$ and starting frequencies $\overline{f}_{\overline{\theta},p,m,t,i}$, as well as the relative order between those that share common links $\overline{\delta}_{\overline{\theta},p,m,t,i,\overline{\theta'}p',m',t',i'}$. These are passed as input to the D-RSA ILP formulation that defines new equivalent variables for the existing demands and introduce constraints to identify how these new variables differ from the previous ones in order to identify the shiftings and reroutings performed. As expected, the ILP formulation also includes related variables for the new demand. Although in the following we will try to give short descriptions for all the symbols used, the reader is also referred to [16] for a more detailed explanation. Compared to [14], the formulation that we present here indexes demands with θ, instead of (s, d).

The objective is to serve all demands (new and old-active) with the lowest cost in terms of the reroutings and shiftings of the active connections. If the algorithm cannot find a feasible solution, meaning that the available spectrum is not sufficient to serve all the demands, the new demand is blocked, and the network remains in its previous state. The ILP formulation is as follows:

Inputs:

$\widehat{\theta}, \overline{\Theta}, \Theta$	New demand, set of existing demands, set of all (new and existing) demands
$Q_{\theta}, Q_{\overline{\Theta}}$	Set of feasible path-tuple pairs for demand $\theta \in \Theta$ or $\overline{\theta} \in \overline{\Theta}$
$W_{\theta,p,t}$	Number of (translucent) parallel connections, required to serve demand θ using path $p \in Q_{s_\theta d_\theta}$ and tuple $t \in T$, that is, using path-tuple pair (p, t)
F_{total}	Number of available spectrum slots
w	Objective weighting coefficient, taking values between 0 and 1. Setting $w = 0$ (or $w = 1$) minimizes solely the shiftings (or reroutings, respectively)
$\overline{x}_{\overline{\theta},p,t}$	The path-tuple pair (p, t) that was used to serve existing demand $\overline{\theta}$
$\overline{f}_{\overline{\theta},p,m,t,i}$	Starting spectrum slot of the transparent connection (p, m, t, i) of existing demand $\overline{\theta}$ [sub-path $m \in R_{p,t}$ of translucent parallel connection $i \in \{1, 2, \ldots, W_{\theta,p,t}\}$ of path-tuple pair (p, t)]
$\overline{\delta}_{\overline{\theta},p,m,t,i,\overline{\theta'},p',m',t',i'}$	The relative ordering between established connections with common links. Equals 0 if the starting frequency $f_{\overline{\theta},p,m,t,i}$ for transparent connection (p, m, t, i) of active demand $\overline{\theta}$ is smaller than the starting frequency $f_{\overline{\theta'},p',m',t',i'}$ for connection (p', m', t', i'), of demand $\overline{\theta'}$, i.e. $f_{\overline{\theta},p,m,t,i} < f_{\overline{\theta'},p',m',t',i'}$. It exists only if sub-paths $m \in R_{p,t}$ and $m' \in R_{p',t'}$ share a link

Variables:

$x_{\theta,p,t}$	Boolean variable, equal to 1 if path-tuple pair (p, t) is used to serve demand θ (existing or the new demand)
$f_{\theta,p,m,t,i}$	Integer variable, equal to the starting spectrum slot of the transparent connection (p, m, t, i) of demand θ
$\delta_{\theta,p,m,t,i,\theta',p',m',t',i'}$	Boolean variable, equal to 0 if the starting frequency $f_{\theta,p,m,t,i}$ for transparent connection (p, m, t, i) of demand θ is smaller than the starting frequency $f_{\theta'p',m',t',i'}$ for connection (p', m', t', i'), of demand θ'
$v_{\overline{\theta}}$	Number of reroutings (on different path or tuple) performed for existing demand $\overline{\theta}$
$h_{\overline{\theta},p,m,t,i}$	Boolean variable, equal to 1 if active demand $\overline{\theta}$ uses the same path-tuple pair and its transparent connection (p, m, t, i) has changed its starting frequency, and equal to 0 otherwise

$z_{\overline{\theta},p,m,t,i}$	Boolean variable, equal to 1 if active demand $\overline{\theta}$ uses the same path-tuple pair and its transparent connection (p, m, t, i) was rerouted (jump over other), and 0 otherwise
$y_{\overline{\theta},p,m,t,i}$	Boolean variable, equal to 1 if active demand $\overline{\theta}$ uses the same path-tuple pair and its transparent connection (p, m, t, i) was shifted, and 0 otherwise
T_{rer}	Total number of reroutings
T_{sh}	Total number of shiftings

ILP formulation

minimize $w \cdot T_{rer} + (1 - w) \cdot T_{sh}$

The following constraints are added to the constraints (1–8) of [16]:

- *Cost function definition:*

$$T_{rer} = \sum_{\overline{\theta}\in\overline{\Theta}} \left(v_{\overline{\theta}} + \sum_{(p,t):\overline{x}_{\overline{\theta},p,t}=1} \sum_{m\in R_{p,t}} \sum_{i=\{1,2,\ldots,W_{\overline{\theta},p,t}\}} z_{\overline{\theta},p,m,t,i} \right) \tag{9}$$

$$T_{sh} = \sum_{\overline{\theta}\in\overline{\Theta}} \sum_{(p,t):\overline{x}_{\overline{\theta},p,t}=1} \sum_{m\in R_{p,t}} \sum_{i=\{1,2,\ldots,W_{\overline{\theta},p,t}\}} y_{\overline{\theta},p,m,t,i} \tag{10}$$

- *Identify the connections that use the same path but different tuple*
  For all $\overline{\theta} \in \overline{\Theta}$, for $(p,\ t) \in Q_{\overline{\theta}}$ such that $\overline{x}_{\overline{\theta},p,t} = 1$

$$v_{\overline{\theta}} \geq \sum_{t'\neq t} W_{\overline{\theta},p,t'} \cdot |R_{p,t'}| \cdot x_{\overline{\theta},p,t'} \tag{11}$$

- *Identify the connections that have changed path*

For all $\overline{\theta} \in \overline{\Theta}$, for $(p,\ t) \in Q_{\overline{\theta}}$ such that $\overline{x}_{\overline{\theta},p,t} = 1$

$$v_{\overline{\theta}} \geq \sum_{p'\neq p} \sum_{t'} W_{\overline{\theta},p',t'} \cdot |R_{p',t'}| \cdot x_{\overline{\theta},p',t'} \tag{12}$$

- *Identify the connections that use the same path-tuple and have changed their starting frequencies*
  For all $\overline{\theta} \in \overline{\Theta}$, for $(p,\ t) \in Q_{\overline{\theta}}$ such that $\overline{x}_{\overline{\theta},p,t} = 1$, for all $m \in R_{p,t}$, and for all $i \in \{1, 2, \ldots, W_{\overline{\theta},p,t}\}$

$$(1 - x_{\overline{\theta},p,t}) + h_{\overline{\theta},p,m,t,i} \geq \frac{f_{\overline{\theta},p,m,t,i} - \bar{f}_{\overline{\theta},p,m,t,i}}{F_{TOTAL}} \tag{13}$$

$$(1 - x_{\overline{\theta},p,t}) + h_{\overline{\theta},p,m,t,i} \geq \frac{\bar{f}_{\overline{\theta},p,m,t,i} - f_{\overline{\theta},p,m,t,i}}{F_{TOTAL}} \tag{14}$$

- *Identify the connections that use the same path-tuple pair and were rerouted (jump over other active connections)*
  For all $\overline{\theta} \in \overline{\Theta}$, for $(p,\ t) \in Q_{\overline{\theta}}$ such that $\overline{x}_{\overline{\theta},p,t} = 1$, for all $m \in R_{p,t}$, and for all $i \in \{1, 2, \ldots, W_{\overline{\theta},p,t}\}$, for all $\overline{\theta'} \in \overline{\Theta}$, for $(p',\ t') \in Q_{\overline{\theta'}}$ such that $\overline{x}_{\overline{\theta'},p',t'} = 1$, for all $m' \in R_{p',t'}$ where m and m' share at least one common link, and all $i' \in \{1, 2\ldots,\ W_{\overline{\theta'},p',t'}\}$,

$$\begin{aligned}&(1 - x_{\overline{\theta},p,t}) + (1 - x_{\overline{\theta'},p',t'}) + z_{\overline{\theta},p,m,t,i} \\ &\quad + z_{\overline{\theta'},p',m',t',i'} \geq \\ &\delta_{\overline{\theta},p,m,t,i,\overline{\theta'},p',m',t',i'} - \bar{\delta}_{\overline{\theta},p,m,t,i,\overline{\theta'},p',m',t',i'}\end{aligned} \tag{15}$$

$$\begin{aligned}&(1 - x_{\overline{\theta},p,t}) + (1 - x_{\overline{\theta'},p',t'}) + z_{\overline{\theta},p,m,t,i} \\ &\quad + z_{\overline{\theta'},p',m',t',i'} \geq \\ &\bar{\delta}_{\overline{\theta},p,m,t,i,\overline{\theta'},p',m',t',i'} - \delta_{\overline{\theta},p,m,t,i,\overline{\theta'},p',m',t',i'}\end{aligned} \tag{16}$$

- *Connections that were shifted*

For all $\overline{\theta} \in \overline{\Theta}$, for $(p,\ t)$ such that $\overline{x}_{\overline{\theta},p,t} = 1$, for all $m \in R_{p,t}$, and for all $i \in \{1, 2, \ldots,\ W_{\overline{\theta},p,t}\}$

$$y_{\overline{\theta},p,m,t,i} \geq h_{\overline{\theta},p,m,t,i} - z_{\overline{\theta},p,m,t,i} \tag{17}$$

We are looking to identify the active connections that were shifted (what we call case 1) or rerouted in a different path (case 2.1) or in the same path (case 2.2), using the same (case 2.2.1) or different (case 2.2.2) tuple. An active connection that has either changed its path or its tuple is considered as rerouted and belongs to case 2.1 or 2.2.1, counted by constraints (11) and (12), respectively. An active connection that uses the same path-tuple pair can be shifted or rerouted, thus belonging to case 1 or case 2.2.1. Constraints (13) and (14) identify these connections (h variables) by checking their starting frequencies. Then constraints (15) and (16) identify the connections that were rerouted (z variables) using the same path-tuple (case 2.2.1). They do that by examining the ordering of the connections (δ variables) to identify the connections jumps and then solving a related set-cover problem to find the minimum number of reroutings to satisfy these jumps, as will be explained in the next subsection. Constraint (17) finds the number of shifting (case 1) by subtracting rerouted connections with the same path-tuple pair (z variables) from connections that have changed their starting frequencies (h variable).

The objective is to minimize a weighted sum of the connections that are rerouted and shifted. The weighting coefficient w controls the relative significance given to these two cost parameters in the optimization function. Values of w close to 0 make the shifting cost the dominant optimization parameter in which case the algorithm is free to reroute as many connections as needed and keep the connections that are established in the same path utilizing the same spectrum

slots. In contrast, values of w close to 1 make the minimization of total reroutings the dominant optimization parameter. In that case, the algorithm serves the connections trying to keep the established connections in the same path and not changing their relative ordering. The new demand is considered as blocked when the algorithm cannot find a feasible solution.

The above ILP algorithm aims to serve each new demand by making the less possible changes to the network. The number of variables and constraints used by the above ILP formulation depends on the overlapping of the paths considered (and thus depends on the topology and the value of k used) as also on the number of active connections that are served each time. In particular, constraints (11) and (12) are employed for each active demand $\overline{\Theta}$. Constraints (13), (14) and (17) are employed for every active connection and not for the connection(s) of the new demand. Constraints (15) and (16) need to be employed for every pair of transparent connections that share at least one common link. In the worst case scenario where we have a set of $\overline{\Theta}$ demands and a total number of $\Delta = \sum_{\bar{\theta}\in\overline{\Theta}} \sum_{(p,t):\bar{x}_{\bar{\theta},p,t}=1} W_{\theta,p,t} \cdot |R_{p,t}|$ possible transparent connections the formulation would require $|\overline{\Theta}| + 3 \times \Delta$ extra variables. It also would require $|\overline{\Theta}|$ inequality constraints for (11) and (12), Δ extra inequality constraints for (13, 14, 17) and Δ^2 inequality constraints for (15, 16).

4.1.1 Reroutings and set-cover problem

To make space for a new connection, already established connections are rerouted or shifted in the spectrum domain. Shifted connections are those that utilize the same path but have different starting frequencies while the relative ordering between the established connections remains the same. In particular, in our formulation, the relative ordering between pairs of connections is described with the related δ variables (for pairs that share at least one common link). Unchanged relative ordering means that there are no "jumps" between the connections, or otherwise there are no changes in the values of the δ variables from one state to the next. On the other hand, rerouted connections that remain in the same path are identified by the changes in the relative ordering, that is changes in the δ variables.

Since we can go from one network state to another by rerouting different combinations of connections, we are interested in identifying the minimum number of such reroutings. So we want to find the minimum number of rerouted connections that yield (cover) the changes in the δ variables. Each time a connection is selected to be rerouted, and depending on its final spectrum placement a set containing the pairs of connections with different relative orderings is created. So, as in the set-cover problem, we want to identify the smallest

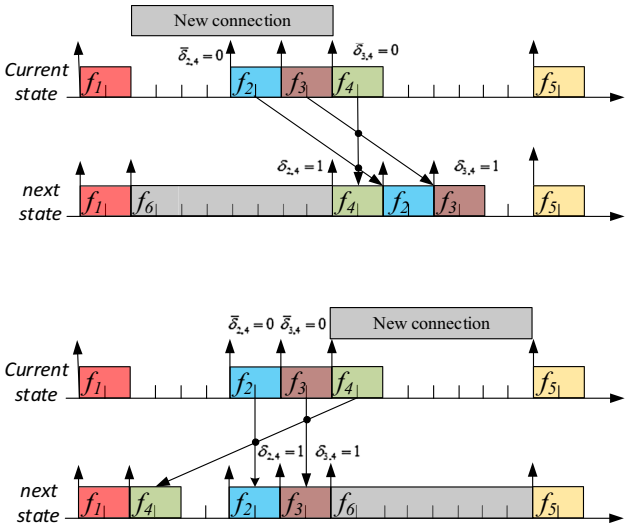

Fig. 2 **a** The spectrum slot allocation on a link at two different states. In the current state, connections 1, 2, 3, 4 and 5 exist and a new connection arrives. Not finding sufficient space to serve it, we reroute connections 2 and 3, and the new connection is established as shown in the next state. **b** The same scenario, with the same changes in the ordering variables, but rerouting only connection 4 to create space for the new connection

subset of S (the smallest number of reroutings) whose union equals the universe (the union of changes in δ values).

Consider the scenario shown in Fig. 2 where we focus on a specific link with several active connections. In order to serve the new connection, there are many different options to rearrange the active connections and make the required space. Some connections may be shifted and some others may be rerouted (remaining in the same path with different relative order). In the example shown in Fig. 2, we take under consideration only the possible reroutings to show how this is related to the set-cover problem. Assuming changes in the ordering of connection pairs (2,4) and (3,4), we want to find the minimum number of reroutings needed to have these changes. In other words, we want to reroute connections that would create changes in δ values that cover the set $U = \{\delta_{2,4}, \delta_{3,4}\}$. This set is our universe containing all the changes in the relative orderings between the connections. To cover this set, we can reroute the connections with starting frequencies f_2 and f_3, as shown in Fig. 2a. However, we can cover all of the elements by rerouting only connection f_4, as shown in Fig. 3b. We would select the second option, since it has the smaller number of reroutings.

Clearly finding the minimum number of rerouted connections, given the changes in the ordering values, is equivalent to the set-cover problem. This observation is taken into account in the proposed ILP formulation, and in particular constraints (13) and (14) formulate a related set-cover problem to identify the minimum number of reroutings to serve the demand and go to the next network state.

(a)
```
INPUTS: Network topology G=(V,E)
        Tuple lookup table
        W:Objective weight
        H:Recursion depth threshold
        Slot utilization vector of links
        New demand (s, d, D,M)

Calculate k paths

FOR each path-transmission tuple pair
  Calculate number of
      connections,spectrum slots needed
  Calculate path slot utilization vector
  Find spetrum voids
  FOR each connection
    IF(size(void)>required_spectrum)
                Select best fit void
                Establish the connection
    ELSE
          Call defragmentation
    ENDIF
  ENDFOR
ENDFOR
```

(b)
```
Push_pull_defrag(connection)

FOR each blocked connection
 Select the biggest void
 Z←Number of slots to push the adjacent connections
 FOR ALL combinations of pushing upper by Zu
       and bottom by Zl Z=Zu+Zl
   Find upper and bottom connections
   Push each connection Z slots
   WHILE (pushing feasible)
       WHILE (more connections to push)
       IF (Recursion_step≤H)
         Call Push_pull_defrag(adjacent_connection)
         WHILE (More connections to push)
           IF (no other combinations exist)
             Select minimum cost combination
           ENDIF
         ENDWHILE
       ENDIF
     ENDWHILE
   ENDWHILE
   Select the combination with the minimum cost
 ENDFOR
ENDFOR
```

(c)
```
Joint_defrag(connection)

FOR each blocked connection
 Select the biggest void
 Fs←starting frequency of connection
 Fe←ending frequency of connection
 Z←number of slots to push the adjacent connections
 FOR ALL combinations of rerouting and shifting upper by Zu
       and bottom by Zl Z=Zu+Zl
  AC'←connections with at least one spectrum slot in the
       interval [Fe, Fe+Zu]
  BC'← connections with at least one spectrum slot in the
       interval [Fs-Zl, Fs]
   For each direction (set of connections)
    N_conn=0
    Reroute the first N_conn connections
    Calculate free space
    IF (enough space)
      Calculate cost
    ELSE
      FOR each connection in (AC' or BC')-rerouted
            connections
        Call Push_pull_defrag(connection)
      ENDFOR
    ENDIF
    N_conn=N_conn+1
   ENDFOR
   Select minimum cost combination
 ENDFOR
ENDFOR
```

Fig. 3 **a** Pseudocode of the algorithm that served the demand by examining all combinations of path-transmission tuple pairs establishing the connections in the existing spectrum voids. **b** Pseudocode of the push–pull algorithm. **c** Pseudocode of the joint (unified) algorithm

4.2 Heuristic algorithm

We now present the heuristic D-RSA algorithm.

The algorithm takes as input the feasible transmission options described by the (reach-rate-spectrum-guardband-cost) tuples, the number of candidate paths k to be checked for each demand, a threshold H on the depth of neighbouring connections that can be pushed or rerouted, and the current state of the network. The current state of the network is described by link spectrum utilization vectors. The spectrum utilization of a link l is represented by a three-state vector U_l, called the link slot utilization vector, of length equal to F_t. We represent by U_{li} the $i-$th slot. A spectrum slot can be in one of the following states: (1) free (denoted by state u_f), (2) used for data transmission (denoted by u_d) or (3) used as guardband (denoted by u_g). The rules are that data slots cannot be used by new connections, free slots can be used for data, while free and guardband slots can be used for guardband by new connections. The slot utilization vector U_p of a path p can be computed using an (associative) 3-ary operator $\oplus$ for combining ("adding") the spectrum slots of the links that comprise it. The combining operator is defined as follows:

$$\begin{aligned} u_f \oplus u_d &= u_d, u_f \oplus u_g = u_g, u_f \oplus u_f \\ &= u_f, u_g \oplus u_g = u_g, \\ u_d \oplus u_d &= u_d, u_d \oplus u_g = u_d \end{aligned}$$

Thus, $U_{pi} = \bigoplus_{l \in p} U_{li}$, for all $i = 1, 2, \ldots, F_t$.

The D-RSA heuristic algorithm serves the demands as they arrive. As we saw in the start of this section a demand depending on the selected path-tuple pair may require the establishment of one or more transparent connections, depending on the regeneration points and the number of parallel breakings. The D-RSA heuristic algorithm establishes these connections, finding spectrum for them and if it cannot do such it defragments the network. To achieve this, we have developed a push–pull and a rerouting algorithm, which can execute alone or combined.

In short, the push–pull algorithm we implemented selects a spectrum void and creates the required space to establish the blocked connection by shifting the void's neighbouring connections. The cost of using this particular void is defined as the number of shifted connections. Other important metrics that can be optimized are the spectrum shifting (in slots) of the longest shifted connection, since this determines the time for performing the push–pull(s) or the total number of slots of all connections that are shifted, etc. The rerouting algorithm we devised reroutes upper and lower neighbouring connections from the selected void so as to free the required spectrum space. It examines all possible combinations of reroutings, similarly to the push–pull algorithm, and selects the combination that minimizes the cost, considered to be the number of rerouted connections. Note that the main difference between the proposed heuristic and the ILP-based algorithm presented in the previous section is that the heuristic selects a specific void and searches to make space around that void, while the ILP algorithm searches among the reroutings and shiftings of all active connections.

To be more specific, to serve the demand with a specific path-tuple pair, the algorithm computes the path utilization vector based on the links that comprise the path. For each connection required to serve the demand with the specific path-tuple pair, the algorithm checks whether there are voids of spectrum able to serve that connection, taking also into account the guardband needed. If there are more than one voids, the algorithm selects the smallest one so as to leave

bigger voids for future connections with higher spectrum needs. If there is no void to accommodate at least one connection of the path-tuple pair under consideration, we move to examine the next path-tuple pair. We stop the first time we are successful with a path-tuple pair, that is, we find appropriate voids to serve all connections required for that path-tuple pair selection. If we examine all candidate path-tuple pairs and none was successful, we proceed with our network re-optimization techniques: push–pull and reroutings. Serving the demands with the available voids is considered zero cost and is preferred over the case where we have to re-optimize the network. The above phase of the algorithm is described in Fig. 3a.

When we have ruled out the option of serving the demand without re-optimizing the network, we again start searching all candidate path-tuple pairs. For each path-tuple pair, for each of its connections that there is not enough spectrum the algorithm selects the biggest void and tries to create the extra slot space in one of the following ways.

Assume that we use the push–pull technique and start with a void that we need to expand by n slots to establish the connection that has guardband needs of gb_v spectrum slots. We can make this spectrum space by shifting connections that are upper or bottom adjacent to the void under examination. Let F_s and F_e be the first free slot and the last free slot of the void, respectively. For an upper connection i, we let F_i be its starting frequency and gb_i be its corresponding guardband needs.

Then we have to shift this connection by

$$n - Fi + F_e - \max(gb_v, gb_i)$$

For a bottom connection j, we again let F_j be its ending frequency and gb_j be its corresponding guardband needs. Then we have to shift this connection by

$$n - F_j + F_s - \max(gb_v, gb_j)$$

Shifting one connection may trigger the shifting of its adjacent connections. This is treated recursively by the same algorithm taking the shifted connection as the void. To make the required space, there are $n + 1$ combinations: shift the upper connections and make n slots space or shift the upper to make $n - 1$ slots and the bottom to make 1 slots space, ..., or shift only the bottom to make n slots space. We examine all different upper-bottom pushing combinations and calculate their cost; this is done quite fast as we only examine the two extreme cases, and the costs of the other combinations can be calculated from them. The cost is defined as the number of spectrum slots of the connection that is shifted the longest, since this is proportional to the time required, but other interesting metrics could also be used, including the number of shifted connections or the number of shifted slots of all connections. The algorithm stops execution when there are no other connections that need to be recursively pushed or when we reach the recursion threshold H (see next paragraph) in which case we consider that the connection is blocked for the specific upper-bottom pushing combination. If more than one combinations are feasible, we select the one that yields the smallest cost. If for the given path-tuple pair the void that we examine cannot be expanded, we move to examine the remaining path-tuple pairs that require less spectrum slots. If we examine all path-tuple pair and none is successful, the demand is blocked.

Since shifting an adjacent connection may trigger the shift of its own adjacent connections and so on, this can go deep and we can end up by examining (and making) a huge number of changes in the network, the whole network configuration in the worst case scenario. To avoid the high running time of the algorithm, we use the threshold H to control the recursion depth. The depth of the push–pull algorithm is defined as follows. The initial void has depth zero, and every connection that is shifted inherits the depth level from the connection that shifts it and adds one. Although this limitation increases the blocking probability, it has the advantage of lower running times and can be used in cases when this is a critical parameter. The push–pull algorithm is described in Fig. 3b.

The second way to create the required spectrum space needed to establish the new connection is by rerouting one or more connections. The key difference is that in this case, we re-establish neighbouring connections, and we take them to remote spectrum blocks or over different paths, instead of shifting them. We again start with a void that we need to expand by n slots to establish the connection. We calculate the set of neighbouring connections from each side of the void, denoted by *AC'* and *BC'*, as follows: the set *AC'* contains connections that utilize at least one spectrum slot in the interval $[F_e,\ F_e + n]$, and the set *BC'* contains connections that utilize at least one spectrum slot in the interval $[F_s - n,\ F_s]$. We then try the different combinations of rerouting the connections in these sets. This is done as follows. We first reroute all connections from one side, e.g. the upper side, so as to make spectrum space of n slots. Then we reroute the connections from the upper side so as to create space $n - 1$ slots and reroute connections from bottom side to create space equal to 1 slot, and so on. When a connection cannot be rerouted (we cannot find spectrum to reroute it), the algorithm has to free the remaining slots from the other direction. If connections cannot be rerouted form both directions and the freed spectrum is less than n, then connection establishment is blocked. We search all the different upper-bottom rerouting combinations and calculate for each one its cost defined as the number of reroutings performed multiplied by the number of connections each rerouting consists of (higher than one in case of rerouting a translucent connection). Finally, the algorithm selects the combination with the minimum cost.

Note that when rerouting over a different path, it is not granted that the transmission configuration used originally in the rerouted connection will be feasible (e.g. when the new path is longer). To address this, we treat the rerouted connection in a manner similar to that of the new connection, which is established only if there is a void with the appropriate size. In extreme cases where candidate paths have significant difference in their lengths, more than one connection may be required to be established in order to replace the initial connection. Also in order to be consistent and have end-to-end control rerouting, a translucent connection involves the rerouting of the whole connection instead of its transparent sub-connection that causes the blocking.

The aforementioned push–pull and rerouting techniques can be combined and used together in a unified algorithm to achieve even better performance. The algorithm that combines these two techniques works as follows. After performing a rerouting, following the same direction of the rerouting, the push–pull algorithm is applied to the remaining connections. Then more connections are rerouted, and so on. To be more specific, we again start with a void that we need to expand by n slots to establish the connection. We calculate the set of neighbouring connections from each side of the void, denoted by AC' and BC'. We first try to free the required space from one direction, in our case the upper direction. To do so we start by rerouting the first connection that blocks the expansions and continue shifting the other connections. After calculating the required cost, we start again from the initial state by rerouting the first two connections that block the expansion and shifting the other connections until we create the required space. When we have exhausted all combinations, we continue the same process for creating n-1 slots in the upper side and 1 slot in the bottom side. Note that this case contains also the cases where only shifting or rerouting of the connections is allowed and are also examined as individual algorithms. When we have exhausted all combinations, we continue the same process for creating $n-1$ slots in the upper side and 1 slot in the bottom side. The cost of the solution is the combined cost of the push–pull and rerouting operations. The joint algorithm is shown in Fig. 3c.

The intuition behind the unified algorithm is that when one or more connections block the expansion of the void, because they cannot be shifted, rerouting may be feasible and vice versa, making feasible the establishment of connections where the two algorithms separately would fail.

5 Numerical results

We implemented the proposed D-RSA algorithms in Matlab and used it to evaluate the performance of the proposed re-optimization techniques. We used the IBM ILOG CPLEX [17] for ILP solving. Spectrum slots were taken to occupy $F = 12.5$ GHz, while the network supports $F_t = 320$ slots. We assumed the use of a single type of flexible OFDM transponder that transmits up to 400 Gbps. The (reach–rate–spectrum–guardband) tuples used as input to these experiments were obtained from studies on physical layer impairments for optical OFDM networks [6]. Demands at each node are generated according to a Poisson process with arrival rate λ and an exponentially distributed duration with mean $1/\mu = 1$ time unit and destination uniformly chosen among all nodes. We run our algorithms for the establishment of 50,000 connections. We targeted a confidence level of 95 % in our simulations. The confidence interval varies for the different cases. As a matter of fact, taking into consideration the worst blocking probability (as shown in Figs. 6a, b, 7a), for the small network with initial link lengths the confidence interval is 0.29 %, for doubled link lengths the confidence interval is 0.28 %, while for the Telefonica network the confidence interval is 0.34 %.

In all cases, the cost for rerouting a connection is seven times the cost of shifting a connection in the spectrum domain (w=0.875). We select the cost of the reroutings to be seven times the cost of shiftings for two reasons. First, it depicts the higher cost since the rerouting procedure requires the use of an extra spare transponder for each connection and may cause certain traffic disruptions (non equal length paths, etc). Second, we conducted a number of simulation experiments to select the weighting coefficients, and we saw that in the network under examination and for the particular traffic at hand, this coefficient gave the best performance. The demanded rate is drawn from a uniform distribution on the close interval [0,400] Gbps, rounded with a 10 Gbps step. This traffic generation was selected to cover scenarios where the network is quite dynamic and adaptable to edge traffic changes.

5.1 Optimality performance of the heuristic algorithms

We start by examining the optimality performance of the heuristic algorithm, comparing its performance to that of the ILP algorithm in small scale experiments.

We performed these experiments, assuming the six-node network topology shown in Fig. 4. We used two different variations of the network, one with the link lengths shown

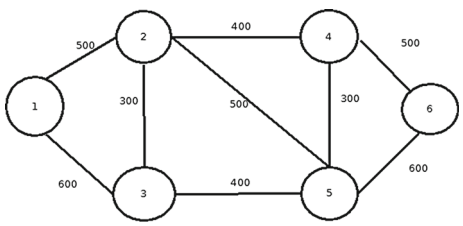

Fig. 4 Small network topology and link lengths in km

in Fig. 4 and one with doubled link lengths. With the initial link lengths, the paths that are created have small lengths and the connections that are established through these paths are only transparent. On the other hand, doubling the lengths of the links results in the establishment of several parallel transparent connections or requires the usage of regenerators in order to make the connections feasible.

For both network cases, the IA-RSA ILP algorithm was able to track optimal solutions at low load. For average arrival rate 2 and higher, the number of established connections increases and the ILP solver was stopped after running for 30 min for each new demand. So, for these cases, we are not sure whether the ILP algorithm found the optimal solutions.

We graph the performance of the D-RSA heuristic with only rerouting technique, only push–pull and both techniques. We also graph the performance of the D-RSA ILP and that of the D-RSA heuristic without re-optimization which are used as upper and bottom bounds, respectively. The performance metrics used are the blocking probability and blocked capacity ratios (weighted blocking probability). The latter metric computes the total requested demand that could not be served because there were not enough free spectrum slots to serve the corresponding connections and is the weight-corrected fraction of blocked connections, with the weights being the capacity of the connections

5.1.1 Network experiments with initial link lengths

In this subsection, we examine the performance of the D-RSA algorithms for the six-node network with the initial link lengths. These lengths are short enough, resulting in the establishment of transparent connections, regardless the transponders configurations, so we will refer to it as transparent network.

For the aforementioned scenario, the rerouting algorithm has blocking probability and blocked capacity ratios close to that of the push–pull algorithm. The six-node network used for our simulation offers limited number of unrelated paths (paths with no common links), reducing the options for the rerouting algorithm which has to reroute a high number of connections in order to make available the required number of spectrum slots.

As expected, the performance of the proposed combined push–pull and rerouting heuristic algorithm is better than the performance of the algorithms which use solely one of these techniques in both metrics (Fig. 5).

With respect to the blocked capacity ratio, the performance between the push–pull and rerouting algorithm is close. The push–pull algorithm seems slightly better for low load, while the rerouting algorithm becomes slightly better at heavy load. At high load, the number of connections that are established

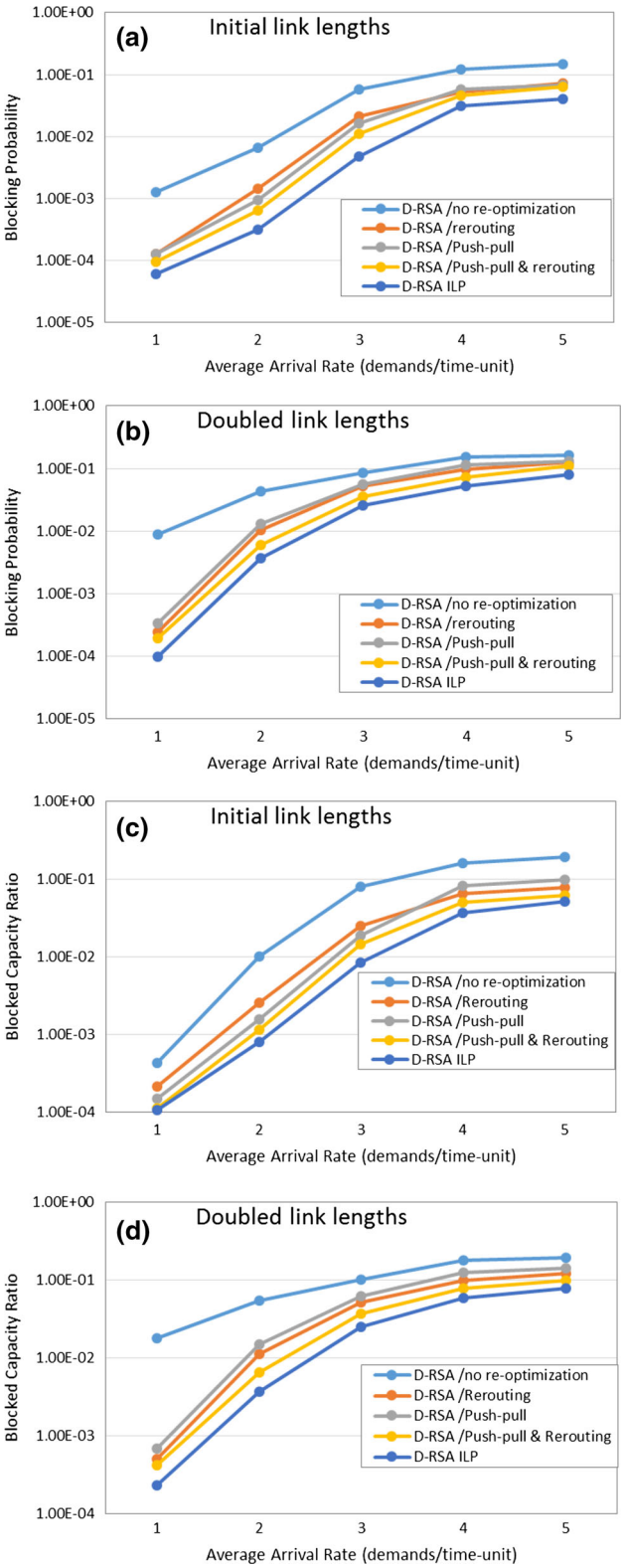

Fig. 5 Blocking probability for the **a** transparent (initial link lengths) and **b** translucent case (doubled link lengths) and average blocked capacity ratio for the **c** transparent and **d** translucent case

in the network is much higher and consequently the number of unused spectrum slots is fewer resulting in more congested links. In such cases, bigger demands cannot be served and are blocked while smaller connections are served, and rerouting is more efficient in avoiding congested links.

Finally, the performance of the heuristic algorithm which combines the push–pull and rerouting techniques for the both evaluation cases is very good and close to that of the ILP algorithm. We can also observe that the proposed defragmentation solutions are effective since they substantially improve the blocking performance when compared an algorithm that does not re-optimize the network.

5.1.2 Network experiments with doubled link lengths

In this case, we assume the same network topology with doubled link lengths. The connections that are established in such a network are mainly translucent requiring in some intermediate nodes the usage of regenerators. As a matter of fact, the transparent connections that are established traverse fewer links relaxing the spectrum continuity constraint of the RSA. So, the network has a higher number of connections of smaller lengths, making easier slot assignment. This explains why when comparing the performance between the two networks, we notice that both the blocking probability and blocked capacity ratios are higher in many cases for the transparent case for the same average traffic.

The performance of the rerouting and push–pull algorithm is very close, with the performance of the rerouting algorithm being slighter better. Since the spectrum continuity constraint is relaxed at the regeneration points, the push–pull technique becomes more efficient, and thus it has slightly better performance than in the transparent case. The joint algorithm has again good performance, close to the ILP, while the no re-optimization algorithm is again much worse than the proposed solutions.

5.1.3 Re-optimization cost

In Fig. 6, we graph the average re-optimization cost required to serve the demands, averaged by the total number of demands that are established (not blocked). The results correspond to the small network topology with the initial lengths. The cost for rerouting a connection is taken seven times the cost of shifting a connection in the spectrum domain ($w = 0.875$). The D-RSA rerouting algorithm has higher cost than the push–pull algorithm, but it needs to shift more connections than the rerouted connections. The cost of the ILP algorithm is much higher than the combined heuristic algorithm. This is the trade-off for the better blocking probability that it offers versus the heuristic algorithm.

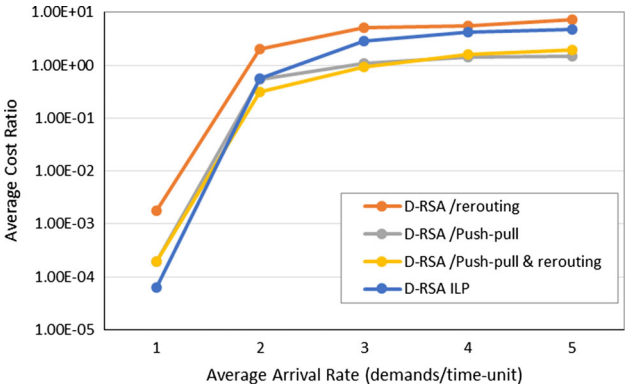

Fig. 6 Average ratio cost comparison

B. D-RSA algorithm performance in a realistic network

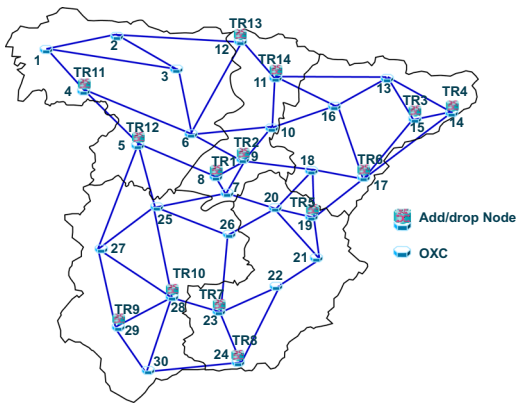

Fig. 7 Reference network based on Spanish national backbone

5.2 D-RSA algorithm performance in a realistic network

In this subsection, we present the performance of the proposed D-RSA heuristic in translucent mode of operation for the Telefonica transport network (Fig. 7).

Figure 8a shows the blocking probability of the D-RSA translucent algorithm, and Fig. 8b shows the blocked capacity ratio for the following algorithms, respectively, (1) without re-optimization, (2) using the push–pull technique, (3) using the rerouting technique and (4) using both push–pull and rerouting (joint). The D-RSA without re-optimization has the worst performance and, as in the previous case, is used as a point of reference of the proposed heuristic algorithms. Both push–pull and rerouting techniques, when used independently, improve the performance in means of blocking probability and blocked capacity ratio, with the performance of the latter being superior in all cases. Specifically, as time passes and links get congested, the new connections that arrive cannot be served. The push–pull technique cannot achieve significant improvements at medium and high load, since the links in the network are congested, not leaving enough free spectrum that can be used for defragmentation.

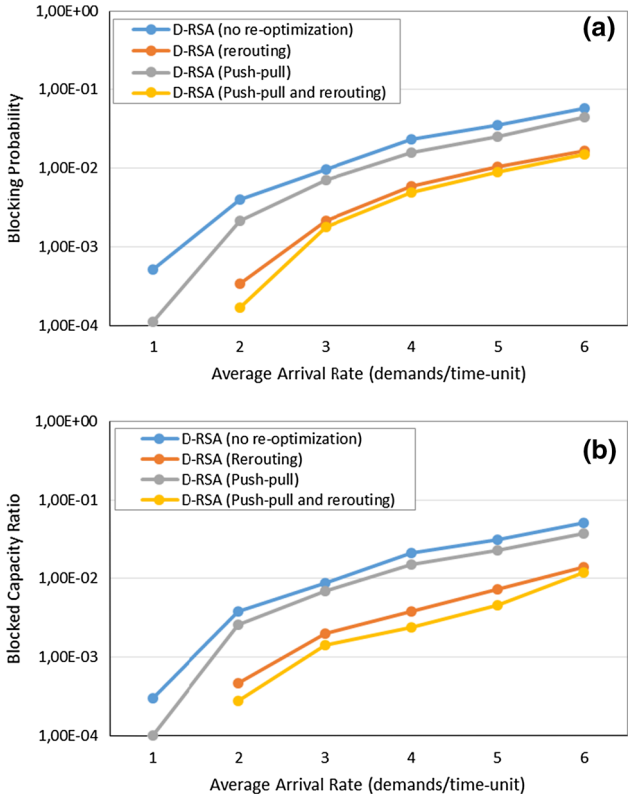

Fig. 8 **a** Blocking probability and **b** blocked capacity ratios for the translucent case of the Telefonica network

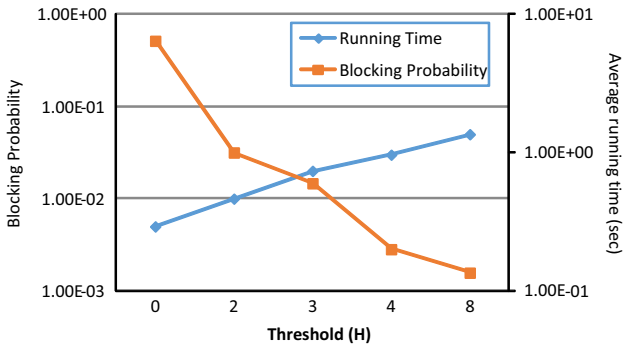

Fig. 9 Blocking probability and average running time of the D-RSA/push–pull and rerouting algorithm for $\lambda = 3$ demands/time unit as a function of the recursion threshold H

However, at lower load, the push–pull technique is quite efficient. Note that network operators would never operate at a load where blocking is high and so the push–pull technique is considered efficient at the network operation point. The rerouting technique has the advantage of exploiting the solution space of different paths, which was shown to be particularly useful at both high and low load. As expected, the algorithm that uses these techniques jointly improves even more the performance, with a slight improvement over the rerouting technique. Thus, the joint algorithm takes advantage of the rerouting technique and the different paths that it exploits, but also at certain instances it uses shifting (push–pull) to create the required space.

Regarding the performance in terms of blocked capacity ratio, the joint algorithm again performs better, with the rerouting algorithm, as previously, being close. Regarding the push–pull technique, we observe that its blocked capacity ratio performance is close to that of no re-optimization, closer than the related blocking probabilities. The reason is that because the push–pull technique tries to create the appropriate space by shifting connections in the spectrum domain, it manages to serve the smaller connections and not the bigger ones that contribute relatively more to blocked capacity ratio metric.

Figure 9 shows the effect of the recursion threshold H on the performance of the D-RSA/push–pull and rerouting algorithm (note that up to now H was set to infinite) for average arrival rate $\lambda = 3$ demands/time unit and translucent network. We see that lowering H increases the blocking probability but reduces the average and maximum running time, introducing a trade-off between these two metrics. The blocking and running time performance for $H > 8$ is similar to that where H was set to infinite. On the other end, when H is set to 0, the performance of the D-RSA/push–pull & rerouting emulates that of the D-RSA/no re-optimization. Thus, H can be chosen so as to meet the response time requirements for serving the demands.

6 Conclusions

We proposed an algorithm for setting up connections and re-optimizing a flexible optical network. When a connection is blocked due to spectrum fragmentation, the push–pull or/and rerouting techniques are used to make appropriate spectrum space with no service interruption. We devised appropriate algorithms that search among different combinations of shifting and rerouting alternatives. Our results show that the rerouting technique seems to be the appropriate approach irrespective of the network load. Its performance can be further improved when assisted by the push–pull technique in a joint algorithm. The push–pull technique is not very efficient at high load, since links are congested and not enough free spectrum is available, but is quite better at lower load. As a matter of fact, both techniques are worth implementing since real networks would not operate at high load with high blocking probability. As our results indicate, we can substantially reduce blocking using the proposed algorithms and we can also trade off performance for running time by appropriate parameter selection depending each time on the offered load and the special characteristics of the network.

Acknowledgments This work has been partially funded by IDEAL-IST Project.

References

[1] Gerstel, O., et al.: Elastic optical networking: A new dawn for the optical layer? IEEE Commun. Mag. **50**(2), S12–S20 (2012)
[2] Gringeri, S., et al.: Flexible architectures for optical transport nodes and networks. IEEE Commun. Mag. **48**(7), 40–50 (2010)
[3] Cugini, F., et al., P.: Push-pull defragmentation without traffic disruption in flexible grid optical networks. Lightwave Technol. J. **31**, 125–133, (2013)
[4] Patel, A.N., et al.: Defragmentation of transparent flexible optical WDM (FWDM) networks. Optical Fiber Communication Conference (2011)
[5] Wang, Xi, et al.: A hitless defragmentation method for self-optimizing flexible grid optical networks. European Conference and Exhibition on Optical Communications (ECOC), Amsterdam, pp. 1–3 (2012)
[6] Eira, A., et al.: Defragmentation of fixed/flexible grid optical networks. Futur. Netw. Mob. Summit, Lisboa, pp. 1–10 (2013)
[7] Luo, Jie, et al.: Partial defragmentation in flexible grid optical networks. Communications and Photonics Conference (ACP), (2012)
[8] Takagi, T, et al.: Disruption minimized spectrum defragmentation in elastic optical path networks that adopt distance adaptive modulation. European Conference and Exhibition on Optical Communications (ECOC), Geneva, pp. 1–3 (2011)
[9] Klinkowski, M., et al.: Elastic spectrum allocation for time-varying traffic in flexGrid optical networks. IEEE J. Sel. Areas Commun. (JSAC) **31**, 26–38 (2013)
[10] Castro, A., Velasco, L., Ruiz, M., Klinkowski, M., Fernández-Palacios, J.P., Careglio, D.: Dynamic routing and spectrum (re) allocation in future flexgrid optical networks. Elsevier Comput. Netw. **56**, 2869–2883 (2012)
[11] Gifre, Ll, Paolucci, F., Aguado, A., Casellas, R., Castro, A., Cugini, F., Castoldi, P., Velasco, L., López, V.: Experimental assessment of in-operation spectrum defragmentation. Springer Photonic Netw. Commun. **27**, 128–140 (2014)
[12] Klekamp, A., et al.: Limits of spectral efficiency and transmission reach of optical-OFDM superchannels for adaptive networks. IEEE Photonics Technol. Lett. **23**(20), 1526–1528 (2011)
[13] Borkowski, R., et al.: Experimental study on OSNR requirements for spectrum-flexible optical networks. J Opt. Commun. Netw. **4**(11), B85–B93 (2012)
[14] Idealist deliverable: D1.1 - Elasticoptical network architecture: reference scenario, cost and planning
[15] Papadimitriou, C., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover publications, NY (1998)
[16] Christodoulopoulos, K., Soumplis, P., Varvarigos, E.: Planning flexible optical networks under physical layer constraints. IEEE/OSA J. Opt. Commun. Netw. **5**(11), 1296,1312 (2013)
[17] IBM Cplex http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

Polizois Soumplis graduated in 2010 from the University of Patras and was awarded the Diploma of Computer Engineer and Informatics. In 2012, he received his M.Sc. degree in computer science and engineering. He is currently pursuing a Ph.D. degree. His research interests are in the areas of network optimization and optical networks.

Konstantinos Christodoulopoulos received the Diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 2002, the M.Sc. in Advance Computing from Imperial College of London, UK, in 2004, and the Ph.D. from Computer Engineering and Informatics Department, University of Patras, Greece, in 2009. He is a post-doc researcher at Trinity College Dublin, Ireland, working on a joint project with IBM. His main research interests are in the areas of algorithms and protocols for optical networks and grid computing.

Emmanouel Varvarigos received a Diploma in Electrical and Computer Engineering from the National Technical University of Athens in 1988, and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 1990 and 1992, respectively. He has held faculty positions at the University of California, Santa Barbara (1992–1998, as an Assistant and later an Associate Professor), and Delft University of Technology, the Netherlands (1998–2000, as an Associate Professor). In 2000, he became a Professor at the department of Computer Engineering and Informatics at the University of Patras, Greece, where he heads the Communication Networks Lab. He is also the Director of the Network Technologies Sector (NTS) at the Research Academic Computer Technology Institute (RA-CTI), which through its involvement in pioneering research and development projects has a major role in the development of network technologies and telematic services in Greece. Professor Varvarigos has served in the organizing and program committees of several international conferences, primarily in the networking area and in national committees. He has also worked as a researcher at Bell Communications Research and has consulted with several companies in the USA and in Europe. His research activities are in the areas of protocols for high-speed networks, ad hoc networks, network services, parallel and distributed computation and grid computing.