Reoptimizing Network Slice Embedding on EON-enabled Transport Networks

Sepehr Taeb*, Nashid Shahriar†, Shihabur R. Chowdhury§, Massimo Tornatore‡, Raouf Boutaba*, Jeebak Mitra§, and Mahdi Hemmati§

*David R. Cheriton School of Computer Science, University of Waterloo, {staeb | rboutaba}@uwaterloo.ca

†Department of Computer Science, University of Regina, Nashid.Shahriar@uregina.ca

‡Politecnico di Milano, massimo.tornatore@polimi.it

§Huawei Technologies Canada Research Center, {shihabur.rahman.chowdhury | jeebak.mitra | mahdi.hemmati}@huawei.com

***Abstract*—5G transport networks will support dynamic services with diverse requirements through network slicing. Elastic Optical Networks (EONs) facilitate transport network slicing by flexible spectrum allocation and tuning of transmission configurations such as modulation format and forward error correction. A major challenge in supporting dynamic services is the lack of a priori knowledge of future slice requests. In consequence, slice embedding can become sub-optimal over time, leading to spectrum fragmentation and skewed utilization. This in turn can block future slice requests, impacting operator revenue. Therefore, operators need to periodically re-optimize slice embedding for reducing fragmentation. In this paper, we address this problem of re-optimizing network slice embedding on EONs for minimizing fragmentation. The problem is solved in its splittable version, which significantly increases problem complexity, but offers more opportunities for a larger set of re-configuration actions. We employ simulated annealing for systematically exploring the large solution space. We also propose a greedy algorithm to address the practical constraint to limit the number of re-configuration steps taken to reach a defragmentated state. Our extensive simulations demonstrate that the greedy algorithm yields a solution very close to that obtained using simulated annealing while requiring orders of magnitude lesser number of re-configuration actions.**

I. Introduction

Transport networks are evolving to support dynamic and short-lived services with diverse quality of service (QoS) requirements [1]–[3], such as enhanced mobile broadband and ultra-reliable low-latency communication [4] through 5G network slicing [5]. This evolution towards network slicing is being fueled by recent advances in Elastic Optical Network (EON) virtualization [5], [6], as EONs support finer-grained spectrum allocation and tuning of transmission configurations (such as modulation format, forward error correction (FEC) overhead, and baudrate) that allow rightsize EON resource allocation to network slices [7], [8]. Hence, EON-enabled transport networks can leverage their flexible resource allocation capabilities to offer to customers tailored and dynamic network slices, typically in terms of a virtual networks (VN) consisting of virtual nodes connected by virtual links.

A key challenge in supporting dynamic services with short lifetime through network slicing is how to deal with the lack of a priori knowledge of slice arrivals and departures. Without such knowledge, it has been observed that slice resource allocation become sub-optimal over time, and, in the specific case of EONs, underlying spectrum resources become fragmented and capacity utilization becomes skewed [9], [10]. Such imbalanced and fragmented spectrum utilization can result in blocking future VN requests [9]. For these reasons, it is essential to re-optimize resources allocated to VNs to minimize fragmentation, ensuring optimal resource utilization, and making room for future VN requests.

In this paper, we address the problem of re-optimizing network slice or VN embedding on an EON. VN embedding on EON involves allocating spectrum resources to a VN by mapping the virtual nodes and virtual links on EON nodes and paths with appropriate transmission configuration [7]. Among the possible objectives, we focus on reducing spectrum fragmentation through re-optimization of VN embedding. Spectrum fragmentation is a direct side-effect of dynamic arrival and departure of VNs and can have far reaching impact such as rejection of future VN establishment requests, resulting in lost revenues. Therefore, defragmentation is considered crucial for network operators and has attracted remarkable attention in research [9]. Note that minimizing fragmentation alone without any consideration for spectrum usage may lead to unbounded increase in spectrum usage. Therefore, measures have to be taken to limit additional spectrum usage during defragmentation, which adds additional challenge to the re-optimization problem. While solving the network slice embedding re-optimization problem, we only consider modifying virtual link embedding. Virtual nodes of a network slice typically have location constraints and host the network functions of a slice. These nodes are usually mapped to an EON node equipped with compute resources located at a metro data center (MDC) or a Point-of-Presence (PoP) site, which are connected by the EON-enabled transport network. EONs in this way form the backbone of telecommunications infrastructure and are primarily concerned with spectrum resources.

We significantly differ from the research literature in optical network defragmentation on several aspects. First, we consider the possibility of splitting a virtual link demand on multiple continuous and contiguous spectrum allocations (called splits) over one or more substrate EON path(s) (similar to [8], [11]). Such splitting facilitates new re-configuration actions for defragmentation, which were not considered in the previous studies. For instance, one split of a virtual link

978-3-903176-36-2 ©2021 IFIP

can further be divided to fill up gaps in spectrum allocation. Similarly, two splits can be merged into a larger split to reduce fragmentation. Second, we do not assume the existence of any special technology (*e.g.*, push-pull [12] and hop-retuning [13]) for performing disruption free defragmentation. We assume the capabilities of commercial transponders and adopt *make-before-break* [14] approach whenever possible.

We quantify spectrum fragmentation using the root mean square fragmentation (RMSF) metric [15], which is non-linear. Due to the non-linearity of our objective function of RMSF, it becomes infeasible to use standard integer/mixed-integer programming modeling tools to obtain optimal solution even for very small problem instances. Therefore, we leverage Simulated Annealing (SA) search [16] to systematically explore a large solution space for obtaining an estimation of a loose lower bound of RMSF. One advantage of using SA is that the search process naturally yields the sequence of actions that must be taken to bring the network to the computed re-optimized state. However, it may arrive at the final defragmented state by applying a excessively large number of re-configuration actions on the network, which might be undesirable from a network operator's perspective. To address this issue, we also devise a greedy search heuristic that navigates through the solution space with a given budget on the number of possible actions. To the best of our knowledge, this aspect of defragmentation was not studied before in the context of splitting-enabled EON. Our extensive evaluation using real network topologies demonstrate that the greedy search process reduces network defragmentation nearly to the same extent as the SA search while applying orders of magnitude lesser number of reconfiguration actions.

The rest of the paper is organized as follows. We first discuss the related works in Section II. Then, we discuss out choice of fragmentation metric and re-configuration actions for defragmentation followed by a formal problem definition in Section III. Then, we present our SA algorithm for re-optimizing VN embedding over an EON in Section IV. Our greedy search approach with a bounded number of re-configuration actions is presented in Section V. Then, we present the evaluation results in Section VI. We conclude with some future research directions in Section VII.

II. Related Works

EON spectrum defragmentation strategies fall into two broad categories, namely, proactive and reactive [9]. Proactive approaches are executed periodically or when fragmentation level reaches a pre-defined threshold [17]. In contrast, reactive approaches defragment spectrum resources when the embedding of a new VN is blocked due to fragmentation and can make room for the newly arrived VN [18]. However, there is no guarantee that reactive defragmentation will be successful and it may result in sub-optimal embedding for the VNs that are migrated during the process. Another line of work for EON defragmentation is fragmentation-aware VN embedding [19], [20]. Fragmentation awareness is achieved by embedding new VNs in a way to reduce spectrum fragmentation. Fragmentation-aware embedding may not be sufficient to defragment an entire EON since such embedding only concerns fragmentation around the new VN. In contrast, proactive defragmentation accounts for all VNs and EON links, and offers more opportunities to re-optimize VN embeddings. Hence, in this paper, we focus on proactive defragmentation by changing the route and/or the spectrum allocation of connections. While doing so, existing approaches assume the availability of techniques, such as push-pull or hop re-tuning, that promises to reconfigure spectrum allocation without causing any traffic disruption [21], [22]. However, these technologies are still experimental and are not available in commercial equipment. Conversely, we do not assume any specific technology for eliminating disruption and adopt the make-before-break approach whenever possible [14].

Shakya *et al.*, proposes a defragmentation approach in [23] that minimizes the maximum used spectrum slot index on any EON link. This objective may not be useful since many fragmented spectrum blocks can exist on a link even when maximum slot index is low. Dávalos *et al.*, presents two metaheuristic-based algorithms for selecting lighpaths that need to be reconfigured for defragmentation [24]. This approach does not generate the order in which lightpaths should be re-configured that we address in this paper. Comellas *et al.*, [17] evaluated several ordering strategies for reconfiguring lightpaths for defragmentation. However, a deterministic order can get stuck at a local minima, mandating to include randomness in the deframengtation algorithm.

Zeng *et al.*, presents an SA based defragmentation mechanism for EONs that modifies the route and/or spectrum assignment of a randomly chosen connection [25]. The cost function in [25] prefers a state where lightpaths have shorter lengths, and spectrum is allocated from a lower frequency. Conversely, we adopt a comprehensive cost function accounting for multiple factors such as number of fragments, size of fragments, and locations of fragments in the spectrum. We also employ several new re-configuration operations and additional optimizations not considered in [25]. One caveat of SA algorithm, both ours and that in [25], is that it requires a large number of steps to reach a re-optimized state, making it impractical. Therefore, we also propose an effective algorithm that reaches a defragmented state in a limited amount of steps, which makes our approach ready to be applied in practice.

III. Problem Definition

A. Problem Statement

We are given an EON G and a set of VNs $\mathcal{G}$ embedded on G. Each VN $\bar{G} \in \mathcal{G}$ consists of a set of virtual nodes (VNodes) $\bar{V}$ and virtual links (VLinks) $\bar{E}$ where each VLink $\bar{e} \in \bar{E}$ has a bandwidth demand $b_{\bar{e}}$. Each VN is embedded by mapping each of its VNodes to an EON node and each of its VLinks to a set of splits with a maximum of q splits. Each split represents a path in the EON where each path p is configured with a transmission configuration represented by a tuple $t = (d, b, m, f) \in \mathbb{T} = (\mathbb{D} \times \mathbb{B} \times \mathbb{M} \times \mathbb{F})$ to provide a data-rate so that the sum of data-rates is $b_{\bar{e}}$. Here, d, b,

TABLE I: Different re-configuration actions and corresponding cost

ID	Re-optimization action	Disruption level	Extra Transponder?	Extra Spectrum?
R_1	Move a split on the same path with the optimal tuple in $\mathcal{R}$	Zero	No	No
R_2	Move a split to a different path with the optimal tuple in $\mathcal{R}$	Zero	No	No
R_3	Merge two splits of the same VLink into one such that spectrum allocation of the new split does not overlap with the spectrum allocations of previous two splits	Zero	No	No
R_4	Merge two splits of the same VLink into one such that spectrum allocation of the new split overlaps with the spectrum allocation of any of the previous two splits	High	No	No
R_5	Divide a split of a VLink into multiple splits on the same path	Zero	Yes	(Possibly) Yes

m, and f represent *data-rate*, *baud-rate*, *modulation format*, and *FEC* selected from the set of possible values $\mathbb{D}$, $\mathbb{B}$, $\mathbb{M}$, and $\mathbb{F}$, respectively. Each tuple t has a spectrum requirement and a maximum optical reach within which t can be used with satisfactory signal quality as defined by a reach table $\mathcal{R}$ [26]. The reach table $\mathcal{R}$ contains a set of tuples each of which has a specific spectrum requirement and a maximum optical reach. Note that the same SPath can be used multiple times as the splits of a VLink following the reasoning in [8]. Also note that $\chi_{\bar{e}i} = (p, t, s_b, s_t) | 1 \leq i \leq q$ represents the i-th split, where $\chi^{(p)}_{\bar{e}i}$ and $\chi^{(t)}_{\bar{e}i}$ denote the selected SPath and transmission configuration for the i-th split, respectively. The spectrum slot allocation for the i-th split begins at index $\chi^{(s_b)}_{\bar{e}i} \in S$ and ends at index $\chi^{(s_t)}_{\bar{e}i} \in S$ along each SLink in the SPath $\chi^{(p)}_{\bar{e}i}$ to satisfy the spectrum contiguity constraint [9].

The VN embedding re-optimization problem seeks to find a feasible sequence of re-configuration actions applied on the VN embeddings such that the resulting embeddings of the VNs optimize (*i.e.*, maximize or minimize) a certain objective function. In our problem, the objective is to minimize network-wide RMSF metric presented in (1). This metric leverages RMSF values of each EON link computed using (2), which is the most comprehensive fragmentation metric [15]. We assume VNode mapping to remain unchanged during the re-optimization process (typical assumption in optical network virtualization [27]). An important question regarding pro-active defragmentation is when to start the re-optimization process that is a problem of its own [22] and out of scope of this paper. In this work, we assume that re-optimization is triggered pro-actively in a periodic manner, or when specific thresholds are exceeded (*e.g.*, fragmentation is high).

$$F^{RMSF}_{net} = \frac{\sum_{e \in E} F^{RMSF}_{e}}{|E|} \frac{s^{max}}{|S|} \tag{1}$$

$$F^{RMSF}_{e} = \frac{s^{max}_{e} |I|}{\sqrt{\frac{\sum_{i \in I} f_i^2}{|I|}}} \tag{2}$$

During re-optimization, the embedding of a VLink of the existing VNs in $\mathcal{G}$ can be re-configured one or multiple times to reduce spectrum fragmentation. VLink re-configuration consists in changing the VLink's embedding *i.e.*, one or more splits' substrate path ($\chi^{(p)}_{\bar{e}i}$), tuple ($\chi^{(t)}_{\bar{e}i}$), and spectrum allocation ($\chi^{(s_b)}_{\bar{e}i}$ and/or $\chi^{(s_t)}_{\bar{e}i}$) or a combination thereof. A re-configured split still has to satisfy spectrum contiguity and optical reach constraints and total number of splits after re-optimization cannot be more than q. We use 5 different re-configuration actions with different levels of disruption and their impact on resource usage (*e.g.*, transponder or spectrum usage) shown in Table I. For instance, R_1 in Table I may change any or all of $\chi^{(t)}_{\bar{e}i}$, $\chi^{(s_b)}_{\bar{e}i}$, and $\chi^{(s_t)}_{\bar{e}i}$ of a split while keeping its $\chi^{(p)}_{\bar{e}i}$ fixed, whereas R_2 can change all of them. Splitting offers us new re-configuration actions (R_3, R_4, and R_5) that were not considered in prior work. For example, we can merge two splits into a larger one using R_3/R_4 or divide a large split into multiple smaller ones using R_5.

All actions except R_4 in Table I can be applied using *make-before-break* to avoid any disruption as long as the spectrum allocation after applying any of these actions does not overlap with the previous spectrum allocation. Note that *make-before-break* requires double committed resource for a short period of time when it is applied. There are scenarios when the spectrum allocation after re-configuration overlaps with the allocation present before applying the action such as for action R_4 in Table I. In such scenarios, it might not be possible to apply *make-before-break*, therefore, the disruption level can be significantly high. However, disruptive action such as R_4 may be useful to defragment a highly utilized EON and should be used only when necessary.

B. Pre-computations

For each VLink $\bar{e} \in \bar{E}$, we pre-compute $\mathcal{P}^k_{\bar{e}}$, a set of k shortest paths between each pair of SNodes that a VLink's end VNodes have been mapped to. For each SPath $p \in \mathcal{P}^k_{\bar{e}}$, we pre-compute the set of admissible transmission configurations, $\mathcal{T}_{\bar{e}p} \subset \mathcal{T}$, such that each configuration $t \in \mathcal{T}_{\bar{e}p}$ results in a reach $r_t \geq len(p)$ and has a data-rate $t^{(d)}$. $\mathcal{T}_{\bar{e}}$ contains all the distinct tuples suitable for $\bar{e}$ and is defined as $\bigcup_{\forall p \in \mathcal{P}^k_{\bar{e}}} \mathcal{T}_{\bar{e}p}$.

IV. Re-optimization with Unbounded Number of Actions

In this section, we present a Simulated Annealing [16] (SA) based algorithm for solving the VN embedding re-optimization problem. SA allows us to systematically search through a solution space, while maintaining a lineage of operations starting from the initial network state to reach a feasible re-optimized state. The advantage of SA is that it can avoid getting stuck at local minima by employing random moves, and converge to a global minimum if the SA process is run for a sufficiently long period of time.

A. Simulated Annealing (SA) Algorithm

SA systematically explores the neighborhood of an initial set of embeddings of all VLinks in $\mathcal{G}$ (*i.e.*, $current\_state$) and

keeps reducing the fragmentation of the EON G. A *neighbor* to the $current\_state$ is another set of embeddings where the embedding of only one VLink from the $current\_state$ is re-configured using one of the actions presented in Table I. SA then evaluates quality of the neighbor using our objective function of F_{net}^{RMSF} in Eqn. (1) and moves to a neighbor that improves F_{net}^{RMSF}. It also probabilistically accepts a worse neighbor (a re-optimized embedding with a higher value of F_{net}^{RMSF} than the $current\_state$) from the search neighborhood. Typically, a temperature parameter (T) and energy function controls the probability of accepting a worse neighbor. Parameter T is set to a higher value during the initial iterations of the search, resulting in a higher probability of accepting a worse neighbor. A cooling schedule attenuates the temperature and eventually decreases the probability of accepting a worse neighbor towards the end of the SA search. By accepting worse solutions, SA tries to avoid being stuck at a local minimum. We run multiple iterations of the neighborhood exploration procedure for each temperature to cover a wider search space. To better explain the SA algorithm, we define the following:

- A neighborhood generation function, which, generates a set of neighbors to the $current\_state$ and returns a neighbor according to a policy.
- An energy function, which determines the fitness of a neighbor. It regulates the probability of accepting or rejecting a neighbor during an iteration of SA.
- A cooling schedule, which determines how the temperature is attenuated during the search process.

1) Neighborhood Generation: We devise an algorithm (Algorithm 1) for generating a set of neighbors to the $current\_state$ and returning one of the neighbors. Algorithm 1 first chooses a random split Y from the uniformly distributed set of embeddings of the VLinks of $current\_state$. Then, Algorithm 1 generates all feasible neighbors, $all\_neighbors$, by applying each action from Table I on split Y. Note that the number of feasible neighbors for a single action can be large due to the many possible feasible spectrum re-allocation of a split Y. Since there can be a substantial number of feasible neighbors in $all\_neighbors$ and a significant portion of them are not helpful for re-optimization, Algorithm 1 adopts two heuristics for keeping the most promising neighbors in $neighbor\_pool$. First, Algorithm 1 excludes those neighbors from $all\_neighbors$ that use $\delta\%$ additional spectrum slots compared to that used in split Y's current embedding (line 4). We call δ as *Slot Usage Limit*. The rationale for doing so is that a network operator may not want to increase spectrum usage during re-optimization and may want to move to neighbors whose spectrum usage remains within a specified bound. Second, Algorithm 1 populates a sorted list of the remaining neighbors in $all\_neighbors$ in increasing order of the neighbors' F_{net}^{RMSF} values ($sorted\_neighbors$) and adds only the first $\Gamma\%$ (called as *Neighborhood Limit*) neighbors from $sorted\_neighbors$ to $neighbor\_pool$ (line 6). Here, both δ and Γ parameters and can be tuned based on network operator policies.

Algorithm 1: Select-Neighbor

1 **function** *Select-Neighbor*($current\_state, \delta, \Gamma$)
2 Choose a random split Y from $current\_state$
3 $all\_neighbors \leftarrow$ Generate all feasible neighbor states by applying actions from Table I on split Y
4 Exclude neighbor states from $all\_neighbors$ that uses $\geq \delta\%$ additional slots compared to $current\_state$
5 $sorted\_neighbors \leftarrow$ Sort the neighbors in $all\_neighbor$ in increasing order of their RMSF
6 $neighbor\_pool \leftarrow$ Select the first $\Gamma\%$ neighbors from $sorted\_neighbors$
7 $max\_RMSF \leftarrow$ Maximum value of RMSF among the neighbors in $neighbor\_pool$
8 **for** $neighbor \in neighbor\_pool$ **do**
9 $Gain_{neighbor} \leftarrow (RMSF_{neighbor} - max\_RMSF)^2$
10 $< neighbor, action\_spec > \leftarrow$ A neighbor from $neighbors\_pool$ with a probability proportional to $Gain_{neighbor}$ and corresponding action
11 **return** $< neighbor, action\_spec >$

After generating $neighbor\_pool$, Algorithm 1 finds the maximum value of F_{net}^{RMSF} among the neighbors in $neighbor\_pool$. This value is then used to compute the relative RMSF gain ($Gain_{neighbor}$) of each of the other neighbors in $neighbor\_pool$ (line 8 – 9). $Gain_{neighbor}$ is defined as the square of the difference between the neighbor's F_{net}^{RMSF} and the maximum value of F_{net}^{RMSF}. Finally, the algorithm returns a neighbor with the probability proportional to its RMSF gain and $action\_spec$, the action that generated the neighbor.

2) Energy Function and Cooling Schedule: We use our fragmentation metric F_{net}^{RMSF} defined in Eqn. (1) as the energy function. During iteration k of SA search, the probability of moving to a neighbor is a function of energy and temperature [16]. This probability is defined as follows:

$$\mathcal{P}(Energy, T_k) = \begin{cases} 1 & \text{if } \Delta_{Energy} < 0 \\ e^{-\Delta_{Energy}/T_k} & \text{otherwise.} \end{cases}$$

Here, T_k is the temperature at the k-th iteration and $\Delta_{Energy} = F_{net}^{RMSF}(current\_state) - F_{net}^{RMSF}(new\_state)$, where $F_{net}^{RMSF}(current\_state)$ and $F_{net}^{RMSF}(new\_state)$ are energies of $current\_state$ and the neighbor returned by Algorithm 1, respectively. We use a linear cooling schedule [28] and set the temperature T at iteration $k+1$ as: $T_{k+1} = \rho * T_k$, where $0 < \rho < 1$ is the cooling rate (Line 21).

The SA algorithm, as outlined in Algorithm 2, takes as input an initial state, *i.e.*, current set of VN embedding ($\mathcal{C}$), maximum number of iterations to perform (it_{max}), the number of iterations to perform per temperature value (it_{temp}), an initial temperature (T_0) and the cooling rate (ρ). During each iteration for a particular temperature value, new_state and its $action\_spec$ are generated by invoking the Select-Neighbor procedure from Algorithm 1 (line 8). Based on the energy of the new_state and the current temperature, the SA search moves to the new_state or not (line 11 – 15). During the search, SA keeps track of the best state ($best\_state$) and corresponding action sequence ($best\_sequence$) according

Algorithm 2: SA-Re-Optimize

```
1  function SA-Re-Optimize(C, it_max, it_temp, T_0, ρ)
2      iterations ← 0, T ← T_0
3      action_sequence ← best_sequence ← φ
4      best_state ← current_state ← C
5      while iterations < it_max do
6          while iterations_t < it_temp do
7              current_cost ← F_net^RMSF(current_state)
8              < new_state, action_spec > ← Select-Neighbor
                (current_state, δ, Γ)
9              new_cost ← F_net^RMSF(new_state)
10             Δ_energy = current_cost − new_cost
11             p ← rand(0, 1)
12             if Δ_energy < 0 or p < e^(Δ_energy/T) then
13                 current_state ← new_state
14                 current_cost ← new_cost
15                 action_sequence.append(action_spec)
16             if current_cost < best_cost then
17                 best_cost ← current_cost
18                 best_state ← current_state
19                 best_sequence ← action_sequence
20             Increment iterations_t
21         T = ρ * T, Increment iterations
22     return < best_state, best_sequence >
```

to the objective function F_{net}^{RMSF}. Finally, $best\_state$ and $best\_sequence$ generated during all the iterations are returned.

V. Re-optimization with Bounded Number of Actions

Although the SA algorithm from Section IV is capable of systematically exploring a large solution space, it may require an excessively large number of reconfiguration actions to reach the re-optimized state. A long sequence of reconfiguration steps can cause a long period of network instability, rendering the SA algorithm impractical in realistic settings. Another drawback of SA algorithm is the lack of fairness among the involved VLinks, *i.e.*, , some VLinks may be subjected to a significantly more number of reconfiguration actions than others, resulting in longer period of instability for those VLinks. To resolve these two limitations of the SA algorithm, we propose a greedy algorithm that can provide a satisfactory re-optimization performance using a bounded number of total actions and per-VLink actions. In this section, we first discuss the two constraints to bound the number of actions, and then describe how the greedy algorithm enforces these constraints.

Maximum number of actions: This bound defines the maximum number of actions (M_{max}) the re-optimization process can take, similar to the bound proposed in [29]. The solution to the re-optimization problem will generate an ordered sequence of at most M_{max} actions that lead to the best possible re-optimized state.

Maximum number of actions per-VLink: This bound enforces a limit on the maximum number of actions (A_{max}) the re-optimization process can apply on each VLink. Similar bounds have been applied to ensure fairness during defragmentation of wavelength division multiplexed optical networks [29]. Note that one could easily impose a different bound on different VLinks to facilitate a differentiated treatment of VLinks during re-optimization. Such differential treatment can enable a variety of service level agreements where a VLink from the highest priority class is not impacted during re-optimization, while a VLink from best-effort class goes through a substantially large number of re-configurations.

A. Greedy Algorithm

The greedy algorithm presented in Algorithm 3 takes the current set of VN embeddings ($\mathcal{C}$), maximum number of iterations to perform (it_{max}), number of inner iterations (it_{inn}), M_{max}, and A_{max} as inputs. The greedy algorithm follows a similar flow of SA algorithm from Algorithm 2 with three major differences, while ensuring the constraints imposed by the two bounds we discussed. First, Algorithm 3 selects the best neighbor in terms of F_{net}^{RMSF} as opposed to selecting a neighbor chosen based on a probability distribution in Algorithm 2. To do so, Algorithm 3 invokes Best-Neighbor procedure that returns the neighbor with the lowest value of F_{net}^{RMSF} among the neighbors generated by applying the actions from Table I on a randomly selected split Y. Best-Neighbor procedure is a modified version of Algorithm 1 that first picks a random split Y which has been subjected to *Slot Usage Limit* and has not exceeded A_{max} actions, and returns the neighbor with the highest RMSF gain by applying the actions from Table I on split Y. We do not present Best-Neighbor procedure for the sake of brevity. If the neighbor returned by Best-Neighbor procedure improves F_{net}^{RMSF} from the $current\_state$, Algorithm 3 takes the corresponding action and moves to the neighbor state. Since Algorithm 3 selects neighbor in a greedy fashion, it can get stuck to a local minima. To circumvent this issue, Algorithm 2 has a provision to move to a worse neighbor than the $current\_state$ with a low probability as discussed next.

The second difference of Algorithm 3 with Algorithm 2 is that Algorithm 3 moves to a worse neighbor than the $current\_state$ only when it is necessary as opposed to stochastically moving to a worse neighbor in Algorithm 2. This move is triggered when the neighbors of all the splits have worse F_{net}^{RMSF} values than F_{net}^{RMSF} of $current\_state$. This is achieved by incrementing a counter ($it\_counter$) when a worse neighbor is found and comparing $it\_counter$ with total number of splits (no_of_splits) in line 24 – 25. When a better neighbor is found in a particular iteration, $it\_counter$ is reset to start over (Line 12). Finally, Algorithm 3 has a terminating condition when the number of applied actions reaches the threshold M_{max} in line 21. When such condition is reached, Algorithm 3 returns the best neighbor found during the search. Note that we also tried a naive greedy algorithm that chooses a split Y on the EON link with the maximum fragmentation and does not take a worse neighbor at any time. Such a naive greedy algorithm gets stuck to a local minima very soon during the search. Hence, our proposed greedy algorithm blends greedy selection of neighbors with random

Algorithm 3: Greedy-Re-Optimize

```
function Gr-Re-Optimize(C, it_max, it_inn, M_max, A_max)
    iterations ← 0, move_counter ← 0
    action_sequence ← best_sequence ← φ
    best_state ← current_state ← C
    while iterations < it_max do
        it_counter ← 0
        while iterations_t < it_inn do
            current_cost ← F_net^RMSF(current_state)
            < new_state, action_spec > ← Best-Neighbor
              (current_state, δ, Γ)
            new_cost ← F_net^RMSF(new_state)
            if new_cost < current_cost then
                it_counter ← 0
                Increment move_counter
                current_state ← new_state
                current_cost ← new_cost
                action_sequence.append(action_spec)
                if current_cost < best_cost then
                    best_cost ← current_cost
                    best_state ← current_state
                    best_sequence ← action_sequence
                if move_counter = M_max then
                    return < best_state, best_sequence >
            else
                Increment it_counter
                if it_counter > no_of_splits then
                    current_state ← new_state
                    current_cost ← new_cost
                    action_sequence.append(action_spec)
            Increment iterations_t
        Increment iterations
    return < best_state, best_sequence >
```

choice of a split at the beginning and has the provision of selecting a worse neighbor when needed.

VI. Evaluation

A. Simulation Setup

We implement the compared algorithms presented in Section VI-B using C++. We consider a fully-flexible EON using Nobel Germany topology (17 nodes and 26 links) from the SNDlib Repository (available at http://sndlib.zib.de). Each EON link has 4THz spectrum bandwidth divided into 160 slots of 25GHz. To emulate a live EON, we develop a discrete event simulator. The simulator loads the EON with VNs by simulating VN arrival and departure events and allocating and releasing spectrum slots to virtual links accordingly. In our simulator, VN arrival rate follows a Poisson distribution with varying means (20-30 VNs per 100 time units) and VN life time is exponentially distributed with a mean of 100 time units. The number of VNodes vary from 2 to 6 and number of VLinks vary from 3 to 15 (randomly chosen). When a VN arrives, the simulator embeds the VN using the algorithm proposed in [8]. This simulator generates snapshots of the EON's current occupation at different time instances in which a varying number of VNs are embedded. To analyse the performance of the compared algorithms, we take different snapshots of the EON at different utilizations (varying from 30% to 60%). For each network utilization, we pick five random snapshots. For each snapshot, we independently run the compared variants five times and take the output of the run that achieves the best defragmentation. Finally, we report the average (with maximum and minimum as errorbars) of the best performances of each five different snapshots.

B. Compared Variants

1) SA-baseline: This variant represents SA-Re-Optimize algorithm (Algorithm 2) with a huge number of iterations for it_{max} and it_{temp}. The values of $T_0 = 100$ and $\rho = 0.99$ for Algorithm 2 are chosen as best performing ones from multiple trials. This variant also uses *Slot Usage Limit* as $\delta = 10\%$ and *Neighborhood Limit* as $\Gamma = 10\%$ for Algorithm 1 chosen based on trials. *SA-baseline* is used to show convergence of SA-Re-Optimize algorithm, and it provides a feasible lower bound for the re-optimization problem.

2) SA-Re-Optimize: This variant shows the performance of SA-Re-Optimize algorithm (Algorithm 2) with a fixed number of total iterations *i.e.*, $it_{max} = 1000$, $it_{temp} = 1000$ and $T_0 = 100$ and $\rho = 0.99$.

3) Gr-bounded: This variant represents Gr-Re-Optimize algorithm (Algorithm 3). In this case, we set $M_{max} = 500$ and $A_{max} = \infty$ in Algorithm 3. We also vary M_{max} and A_{max} to show sensitivity of Algorithm 3.

4) SA-SOA: This variant represents the scenario where we minimize our cost function F_{net}^{RMSF} (*i.e.*, Eqn. (2)) by using the SA-based defragmentation mechanism presented in [25]. To the best of our knowledge the work in [25] is the closest to SA-Re-Optimize algorithm (Algorithm 2). The mechanism presented in [25] is not specific to any cost function, therefore, we use our cost function instead of theirs for a fair comparison.

C. Performance Metrics

- RMSF Reduction: It is the reduction of fragmentation achieved by an algorithm in terms of RMSF (F_{net}^{RMSF}) from the RMSF of the given snapshot before re-optimization. It is defined as (1 - (the ratio of F_{net}^{RMSF} of the EON after re-optimization to F_{net}^{RMSF} of the EON before re-optimization)). The desired value of RMSF Reduction is 1 but that is not practically achievable. However, the closer RMSF Reduction is to 1, the more defragmentation is achieved through re-optimization (*e.g.*, roughly, RMSF reduction of 0.95 means that 95% fragmentation has been reduced compared to the fragmentation before re-optimization).
- Slot Ratio: It is the ratio between the total spectrum slot usage by the VNs after re-optimization and the total spectrum slot usage by the VNs before re-optimization. Slot Ratio smaller than 1 means that spectrum occupation has decreased after re-optimization.
- Action Count: It is the total number of sequential actions adopted by an algorithm to reach its re-optimized state.
- Number of Actions per VLink: It is the average number of actions applied on each VLink by an algorithm to reach its re-optimized state.

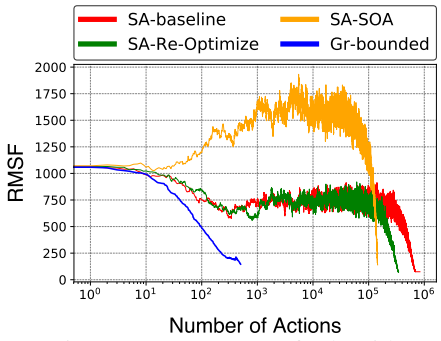


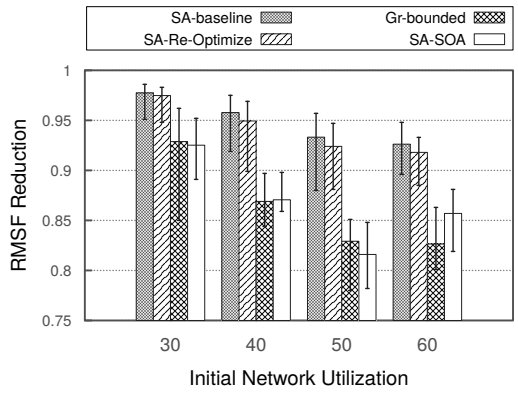
Fig. 1: Convergence of Algorithms

D. Discussion on Evaluation Results

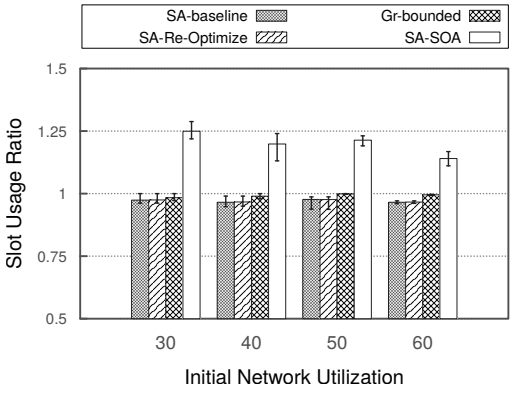
1) Comparison of Different Algorithms: Fig. 1 shows that convergence of the compared variants in terms of RMSF (F_{net}^{RMSF}) values of the EON against number of actions for a specific snapshot with 60% utilization. This figure shows that *SA-baseline* converges to a feasible lowerbound if it is allowed to run for a sufficiently long time. On the other hand, *SA-Re-Optimize* does not converge to a steady value but achieves an RMSF very close to *SA-baseline*'s lowerbound despite being run for a fixed number of iterations. Similarly, *SA-SOA* diverges initially but reaches to an RMSF value closer to the lowerbound through a large number of steps. In comparison to SA based approaches, *Gr-bounded* decreases RMSF of the EON sharply and achieves an RMSF close to *SA-baseline*'s lowerbound even by applying a limited number of actions.

Fig. 2 compares different algorithms in terms of the performance metrics introduced before. In Fig. 2(a), RMSF reduction decreases for larger initial network utilization. This is rational since a highly utilized EON offers less room to re-optimize. Fig. 2(a) also shows that *SA-Re-Optimize* closely approximates *SA-baseline*, and both achieve great performance by reducing more than 90% fragmentation in the given snapshots. On the other hand, *Gr-bounded* and *SA-SOA* achieve lower reductions, yet reductions consistently remain within 10% of *SA-baseline*. *Gr-bounded*, despite using limited number of actions, achieves higher or equal RMSF reduction than *SA-SOA* in all cases except for the snapshots with the highest utilization (60%). Even in the constrained snapshots of high utilization, *Gr-bounded* reduces more than 80% fragmentation in the EON. If not properly designed, defragmentation algorithms might lead to an increase in spectrum slot usage. Our proposed approaches (*SA-baseline*, *SA-Re-Optimize*, and *Gr-bounded*) succeed instead in decreasing slot usage compared to the slot requirement of given snapshots (see Fig. 2(b)) thanks to *Slot Usage Limit* (δ) in Algorithm 1. Note instead that in the case of *SA-SOA*, that does not employ any *Slot Usage Limit*, defragmentation requires more than 20% additional spectrum slots (while this increase is avoided by our algorithms).

Note that RMSF reductions of SA based approaches (*SA-baseline*, *SA-Re-Optimize*, and *SA-SOA*) in Fig. 2(a) require a huge number of re-configuration actions to converge to the most optimized final state (see Fig. 3(a)), in the order of 100's of thousands, which makes their application impractical

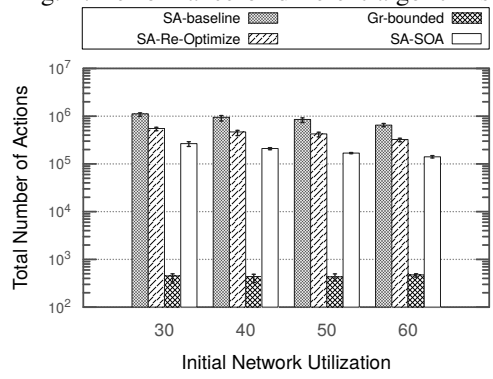


(a) RMSF Reduction

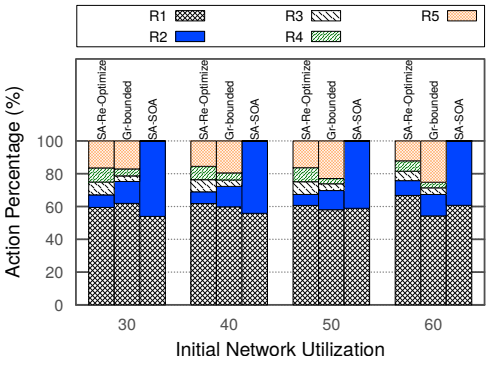


(b) Slot Usage Ratio

Fig. 2: Performance of different algorithms



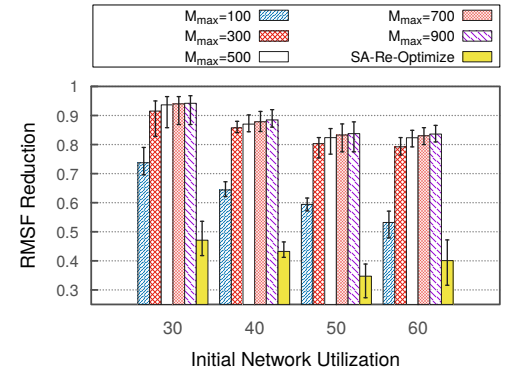
(a) Action Count



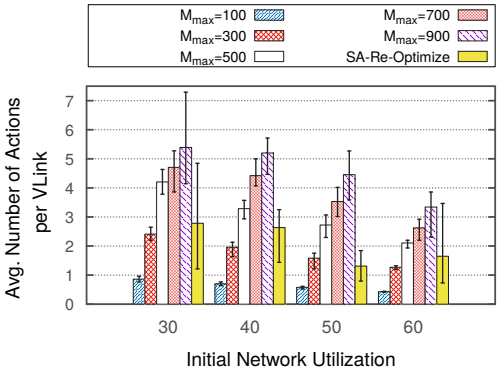
(b) Action Distribution

Fig. 3: Actions taken by different algorithms

in a realistic operational settings. In contrast, *Gr-bounded* achieves more than 80% reduction on fragmentation with no additional spectrum usage by employing only up to 500 re-configuration actions (on average, 2 to 3 actions per VLink), making *Gr-bounded* a more practical solution to be leveraged by a network operator. Finally, Fig. 3(b) shows the distribution of different actions from Table I adopted by *SA-Re-Optimize*, *SA-SOA*, and *Gr-bounded*. According to this figure, usage of action R_1 comprises the major percentages, while the proposed re-configuration actions ($R_3 - R_5$) have been used in more than 20% cases of *SA-Re-Optimize* and *Gr-bounded*. We also observe that the action R_4 which causes disruption has been used very rarely in case of *Gr-bounded*.



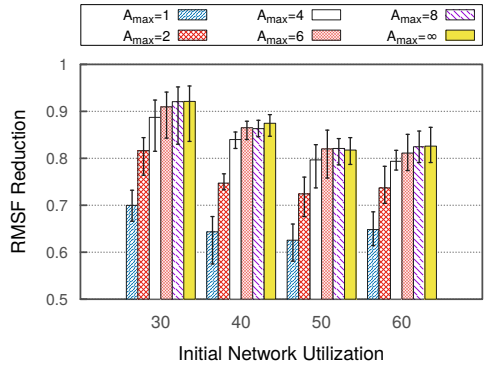
(a) RMSF Reduction



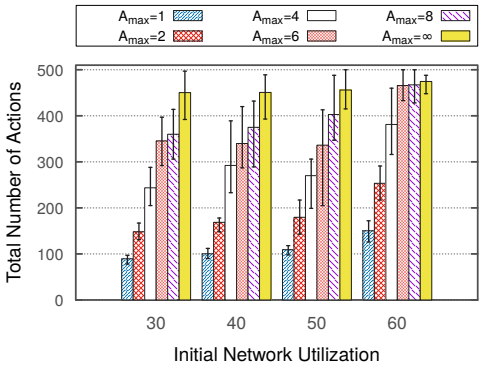
(b) Action Count per VLink

Fig. 4: Performance of *Gr-bounded* by varying M_{max}

2) Analysis of Gr-bounded: Fig. 4 shows the impact of varying M_{max} (Maximum number of actions) on the performance of *Gr-bounded* with $A_{max} = \infty$ and for different values of initial network utilization. Fig. 4 also compares the performance of *Gr-bounded* having different M_{max} bounds with the best possible re-optimized state that could be achieved by *SA-Re-Optimize* in limited number of steps ($M_{max} = 1000$). Fig. 4(a) shows that *Gr-bounded*, even with the smallest M_{max} (M_{max} = 100) achieves 20-30% more defragmentation than the *SA-Re-Optimize* with $M_{max} = 1000$. We also observe that *SA-Re-Optimize* with $M_{max} = 1000$ could not even reduce RMSF more than 50% in any case. The reason for such poor initial performance of *SA-Re-Optimize* is that it allows to move to a worse neighbor with a high probability during the initial stages of SA search that diverges *SA-Re-Optimize* within $M_{max} = 1000$ actions. Such behavior is fundamental to the SA algorithm and, for this reason, *SA-Re-Optimize* cannot be used in a practical setting where the number of actions is limited. Another takeaway from Fig. 4 is that RMSF reduction of *Gr-bounded* increases drastically when the value of M_{max} goes from 100 to 300 and RMSF reduction stabilizes for $M_{max} \geq 500$. This justifies our choice of $M_{max} = 500$ for the other evaluations of *Gr-bounded*. Finally, Fig. 4(b) shows that *Gr-bounded* applies on average one action to each VLink when M_{max} is low ($M_{max} = 100$). For *SA-Re-Optimize*, we keep the best re-optimized state within $M_{max} = 1000$ that can be reached with lower number of actions per VLink. After reaching the best re-optimized state, *SA-Re-Optimize* diverges to worse solutions, and hence we ignore those actions justifying the lower number of per-VLink actions. This figure also shows that average number of actions per VLink increases with the increase in M_{max} and decreases when network utilization increases. This is due to the fact that, as utilization increases, snapshots have more VLinks, and the same total number of actions are distributed among more VLinks thus reducing number of actions per VLink.



(a) RMSF Reduction



(b) Action Count

Fig. 5: Performance of *Gr-bounded* by varying A_{max}

Let us now discuss the impact of varying A_{max} (Maximum number of actions per-VLink) on the performance of *Gr-bounded* with $M_{max} = 500$ and for different values of initial network utilization. Recall from Section V, $A_{max} = x$ means that a VLink can go through at most x number of re-configuration actions during the whole re-optimization process. Fig. 5(a) shows that *Gr-bounded* with $A_{max} = 1$ achieves the lowest RMSF reduction as it allows a single

re-configuration action for all the splits of a VLink. Such a conservative version of *Gr-bounded* still reduces more than 60% fragmentation using the smallest number of actions as shown in Fig. 5(b). As we loosen the constraint of A_{max} (by increasing its value), a VLink can be re-configured more than once offering more opportunities to re-optimize. Hence, RMSF reduction increases for increasing A_{max}. Such gain is more prominent for smaller values of A_{max} while smaller increases in RMSF reduction are observed for $A_{max} \geq 6$. This is due to the fact that the other limit of Maximum number of actions ($M_{max} = 500$) dominates in *Gr-bounded* for $A_{max} \geq 6$. Similar observations apply to the total number of actions in Fig. 5(b). Another takeaway from Fig. 5(b) is that action count increases with the increase in utilization of snapshots. This is expected as a snapshot with higher utilization has more VLinks that requires more actions to be applied.

VII. Conclusion

In this paper, we have addressed the re-optimization of network slice embedding over EON with the objective to minimize spectrum fragmentation. Given an EON with a set of embedded VNs, the problem is to re-optimize the existing VN embeddings by employing a series of different re-configuration actions. We advance the state-of-the-art by addressing, for the first time, the spectrum defragmentation problem in the context of splitting-enabled EON and by proposing novel re-configuration actions that offer more opportunities to re-optimize. Given the complex and non-linear nature of this problem, we have proposed simulated annealing based algorithm for systematically exploring its large solution space. We also propose a greedy search mechanism to address the practical constraint to limit the number of re-configuration steps taken to reach a final defragmentated state.

Our extensive simulation results demonstrate that the simulated annealing based algorithm reduces more than 90% fragmentation at the expense of employing a huge number re-configuration actions. Our results also show that the proposed greedy search algorithm, if we consider a scenario with limited number of re-configuration actions, achieves better defragmentation (reduces more than 80% fragmentation) than a prior approach that employs simulated annealing. The greedy search algorithm can impose different level of bounds on the number of actions to be taken, enabling a fair or prioritized treatment of VLinks during re-optimization. Moreover, our results show that, when the number of admissible reconfiguration action is limited (*e.g.*, below 500), having the possibility to exploit a comprehensive set of reconfiguration actions (involving merging and dividing splits of a virtual link) allows to achieve noticeable improvement in defragmentation. In future, we plan to investigate reactive fragmentation approaches and fragmentation-aware embedding in a splitting-enabled EON.

References

[1] B. Yan *et al.*, "Tidal-traffic-aware routing and spectrum allocation in elastic optical networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 10, no. 11, pp. 832–842, 2018.

[2] M. Hadi, M. R. Pakravan, and E. Agrell, "Dynamic resource allocation in metro elastic optical networks using lyapunov drift optimization," *J. of Opt. Commn. and Net.*, vol. 11, no. 6, pp. 250–259, 2019.

[3] Z. Zhong *et al.*, "Provisioning short-term traffic fluctuations in elastic optical networks," *IEEE/ACM TNet.*, vol. 27, no. 4, pp. 1460–1473, 2019.

[4] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[5] S. Aleksic, "Towards fifth-generation (5g) optical transport networks," in *Proceedings of ICTON*, 2015, pp. 1–4.

[6] R. Boutaba, N. Shahriar, and S. Fathi, "Elastic optical networking for 5G transport," *Springer JNSM*, vol. 25, no. 4, pp. 819–847, 2017.

[7] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *IEEE/OSA Journal of Lightwave Technology*, vol. 32, no. 3, pp. 450–460, 2014.

[8] N. Shahriar *et al.*, "Achieving a fully-flexible virtual network embedding in elastic optical networks," in *IEEE INFOCOM*, 2019, pp. 1756–1764.

[9] B. C. Chatterjee, S. Ba, and E. Oki, "Fragmentation problems and management approaches in elastic optical networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 183–210, 2017.

[10] H. Beyranvand *et al.*, "An analytical framework for the performance evaluation of node-and network-wise operation scenarios in elastic optical networks," *IEEE TComm*, vol. 62, no. 5, pp. 1621–1633, 2014.

[11] A. Pagès *et al.*, "Optimal route, spectrum, and modulation level assignment in split-spectrum-enabled dynamic elastic optical networks," *J. of Opt. Commn. and Net.*, vol. 6, no. 2, pp. 114–126, 2014.

[12] F. Cugini, F. Paolucci, G. Meloni, G. Berrettini, M. Secondini, F. Fresi, N. Sambo, L. Poti, and P. Castoldi, "Push-pull defragmentation without traffic disruption in flexible grid optical networks," *IEEE/OSA Journal of Lightwave Technology*, vol. 31, no. 1, pp. 125–133, 2012.

[13] R. Proietti *et al.*, "Rapid and complete hitless defragmentation method using a coherent rx lo with fast wavelength tracking in elastic optical networks," *Optics express*, vol. 20, no. 24, pp. 26 958–26 968, 2012.

[14] T. Takagi *et al.*, "Disruption minimized spectrum defragmentation in elastic optical path networks that adopt distance adaptive modulation," in *ECOC*, 2011, pp. Mo–2.

[15] P. Lechowicz *et al.*, "Fragmentation-aware algorithm with bordering super-channels in spectrally/spatially-flexible optical networks," *J. of Opt. Commn. and Net.*, 2020, to appear.

[16] E. Aarts and J. Korst, "Simulated annealing and boltzmann machines," 1988.

[17] J. Comellas, L. Vicario, and G. Junyent, "Proactive defragmentation in elastic optical networks under dynamic load conditions," *Photonic Network Communications*, vol. 36, no. 1, pp. 26–34, 2018.

[18] S. Shakya *et al.*, "Virtual network embedding and reconfiguration in elastic optical networks," in *IEEE GLOBECOM*, 2014, pp. 2160–2165.

[19] M. Zhu *et al.*, "Fragmentation-aware vone in elastic optical networks," *J. of Opt. Commn. and Net.*, vol. 10, no. 9, pp. 809–822, 2018.

[20] S. Fernández-Martínez, B. Barán, and D. P. Pinto-Roa, "Spectrum defragmentation algorithms in elastic optical networks," *Optical Switching and Networking*, vol. 34, pp. 10–22, 2019.

[21] R. Wang and B. Mukherjee, "Provisioning in elastic optical networks with non-disruptive defragmentation," *IEEE/OSA Journal of Lightwave Technology*, vol. 31, no. 15, pp. 2491–2500, 2013.

[22] M. Zhang *et al.*, "Dynamic and adaptive bandwidth defragmentation in spectrum-sliced elastic optical networks with time-varying traffic," *J. of Light. Tech.*, vol. 32, no. 5, pp. 1014–1023, 2014.

[23] S. Shakya and X. Cao, "Spectral defragmentation in elastic optical path networks using independent sets," in *National Fiber Optic Engineers Conference*. Optical Society of America, 2013, pp. NTh1I–4.

[24] E. J. Dávalos *et al.*, "Spectrum defragmentation in elastic optical networks: Two approaches with metaheuristics," *IEEE Access*, vol. 7, pp. 119 835–119 843, 2019.

[25] Y. Zeng *et al.*, "Defragmentation of flexible optical networks based on simulated annealing," in *IEEE ACP*, 2012, pp. 1–3.

[26] T. Ahmed *et al.*, "Dynamic routing, spectrum, and modulation-format allocation in mixed-grid optical networks," *J. of Opt. Commn. and Net.*, vol. 12, no. 5, pp. 79–88, 2020.

[27] G. Zhang *et al.*, "A survey on ofdm-based elastic core optical networking," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 65–87, 2012.

[28] S. B. Masti and S. V. Raghavan, "Simulated annealing algorithm for virtual network reconfiguration," in *IEEE Euro-NF Conf. on Next Gen. Internet*, 2012, pp. 95–102.

[29] H. Duong *et al.*, "Efficient make before break capacity defragmentation," in *IEEE HPSR*, 2018, pp. 1–6.