JOURNAL OF
Optical Communications
and Networking

Quality of transmission estimator retraining for dynamic optimization in optical networks

Ankush Mahajan,[1,*] Konstantinos (Kostas) Christodoulopoulos,[2] Ricardo Martínez,[1] Raul Muñoz,[1] and Salvatore Spadaro[3]

[1]Centre Tecnològic de Telecomunicacions de Catalunya, CTTC/CERCA, Castelldefels, 08860, Spain
[2]Nokia Bell Labs, Stuttgart, Germany
[3]Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
*Corresponding author: ankush.mahajan@cttc.cat

Received 30 September 2020; revised 27 December 2020; accepted 28 December 2020; published 9 February 2021 (Doc. ID 411524)

Optical network optimization involves an algorithm and a physical layer model (PLM) to estimate the quality of transmission of connections while examining candidate optimization operations. In particular, the algorithm typically calculates intermediate solutions until it reaches the optimum, which is then configured to the network. If it uses a PLM that was aligned once to reflect the starting network configuration, then the algorithm within its intermediate calculations can project the network into states where the PLM suffers from low accuracy, resulting in a suboptimal optimization. In this paper, we propose to solve dynamic multivariable optimization problems with an iterative closed control loop process, where after certain algorithm steps we configure the intermediate solution so that we monitor and realign/retrain the PLM to follow the projected network states. The PLM is used as a digital twin, a digital representation of the real system, which is realigned during the dynamic optimization process. Specifically, we study the dynamic launch power optimization problem, where we have a set of established connections, and we optimize their launch powers while the network operates. We observed substantial improvements in the sum and the lowest margin when optimizing the launch powers with the proposed approach over optimization using a one-time trained PLM. The proposed approach achieved near-to-optimum solutions as found by optimizing and continuously probing and monitoring the network, but with a substantial lower optimization time.

https://doi.org/10.1364/JOCN.411524

1. INTRODUCTION

An accurate and fast physical layer model (PLM) is required for almost every optimization task of an optical network [1,2]. Today, most optimization tasks are static, such as network setup and upgrading, where calculations are performed in advance. The PLM used includes margins that cover its modeling uncertainties and the evolution of the physical layer conditions over the targeted lifespan [3,4]. Moreover, as soon as the connection is provisioned/established, the vendor can measure its quality of transmission (QoT), e.g., the signal-to-noise ratio (SNR), and correct/improve the configuration. Note that upgrades that involve dynamic operations, such as the establishment of new or reconfiguration of established connections were classified as static above, since typically they are carried out in maintenance windows and not on the operating network. Dynamic reconfigurations for resiliency involve protected/restored connections that were probed beforehand.

In any of these optimization tasks, the PLM needs to be accurate; however, the dynamic operations are not directly applied on the network, an indication of the lack of certainty for such operations. Recently monitoring and machine-learning (ML) techniques have been proposed to account for the actual network conditions and improving the accuracy of the PLM [5–8]. This in turn improves the efficiency of static optimization and paves the way to reduce overprovisioning and realize some dynamic optimization use cases [9–13].

Let us consider a network upgrade/incremental planning task that involves calculations for new establishments and possible reconfigurations of established connections [8,12]. Traditionally a PLM with high margins is used, e.g., considers pessimistic fiber coefficient parameters, full spectral load, or high modeling inaccuracy. The optimization will be quite inefficient and result in considerable overprovisioning. Using monitoring feedback and, e.g., ML [5–8], the parameters of the PLM can be fitted so that its estimated SNR values are close to those monitored in the network. Essentially, feedback and ML are used to understand the current state of the network and increase the PLM estimation accuracy. We will refer to this process as the alignment of the PLM to the physical layer of the network. The PLM accuracy is even more critical when it

is used for dynamic optimization tasks, where the target is to achieve high efficiency in an operating network.

Today, optical networks are moving towards the software-defined networking (SDN) concept, where a centralized controller handles the programmability of all network elements. One of the main advantages of SDN is its intrinsic capability of enabling dynamic optimization operations [14,15]. In this context, the SDN controller implements the optimization logic, interfaces with a PLM, and can be extended to handle closed control loops, which entail the use of monitoring data as input or feedback to conduct the targeted optimization task [7,9,16].

Similar problems arise in almost every industry. To keep up with the rapid advancements of the systems and harvest their improvements in terms of productivity, the digital twin (DT) concept is gaining a great deal of attention. The DT is a digital representation of the real/physical system, used to understand and optimize the targeted system [17]. According to the definition of [18], the DT is more than a model of the system; it includes an evolving set of data, and a means of dynamically adjusting the model. The DT concept was originally introduced in 2003 [19] and first put to public use by NASA [20]. Different industry sectors are taking advantage of DT's ability to simulate real-time working conditions and perform autonomous and intelligent decision-making operations. DT provides an alternative way in today's manual interaction-based design, operation, and service paradigms to solve the related challenges autonomously and in real time [17,19]. Depending on the dynamics of both the system and the optimization process, the DT needs to represent the real system with certain accuracy. To do this, the DT is integrated and realigned with the physical system. Such a realignment mechanism typically involves monitoring and ML schemes.

Turning our attention back to the optical network, the target is to use the PLM as a DT, a model with an appropriate set of parameters and a mechanism to adjust them to support the optimization of the task at hand. For static optimization tasks, such as the incremental planning discussed above, the only option is to train the PLM once, just before making the decisions for the entire optimization task. This results in lower margins and increased network efficiency. But the main target and benefits of DT come in dynamic optimization. In dynamic optimization, we would like to squeeze the margins and achieve higher efficiency, making the accuracy of the PLM a critical factor. For example, the accuracy of the PLM deteriorates as connections are established/released/rerouted/change their power. For dynamic optimization tasks that involve few such calculations and actions, e.g., the establishment or reconfiguration of a single connection, the accuracy of the PLM would be acceptable if it were realigned before the calculation. However, realignment of the PLM is expensive; it requires one or more control loops, including monitoring that can be time-consuming, and thus, it might not be feasible. Moreover, for more complex/multivariable dynamic optimization tasks that require multiple reconfigurations, the accuracy of the PLM can become critical. Algorithms used in such cases are typically iterative; they calculate several intermediate solutions and improve them to find the optimum, which is then configured in the network [11]. However, the accuracy of the PLM deteriorates after several intermediate calculations, and after a point, it can fail to support the optimization calculations. The key advantage is that the network operates, and thus we can realign the PLM/retrain its parameters so that it follows the projections to states intermediately calculated by the algorithm.

In particular, we study the dynamic launch power optimization problem, where we assume that we have a set of established connections and we want to optimize their powers while the network operates. The optimum launch powers can be found with a convex optimization algorithm that performs several intermediate calculations. To solve the problem, three methods are explored: i) having the optimization algorithm probe and monitor the network at each intermediate iteration, ii) using a one-time trained PLM for all optimization iterations, and iii) implementing an iterative closed control loop process that after a number of intermediate iterations configures the network and monitors and retrains the PLM. We will refer to the last option, the proposed solution, as optimization with a DT, since it includes, apart from the PLM, evolving network conditions, appropriate choice of parameters for the PLM, and the process to align it to support the dynamic optimization at hand [18]. Although we applied our proposed solution to the dynamic launch power optimizing problem, the proposed iterative closed control loop that includes the realignment of the PLM is generic. It can be applied to other dynamic multivariable optimization problems, such as dynamic resource allocation, automatic network reconfiguration, defragmentation, and virtual network reconfiguration [9,16,21–25]. It also provides ideas of how to realign the PLM in simpler dynamic and even static optimization tasks.

The remainder of this paper is organized as follows. Section 2 presents an overview of the related work of existing power optimizations schemes, dynamic optimization, and closed control loops. Section 3 presents simulations that expose the optimization mismatch when using a one-time trained PLM with respect to the real world/optical network. Then, in Section 4, we describe the proposed (DT) optimization concept. In Section 5, we evaluate the performance of the proposed scheme. Finally, Section 6 concludes the paper.

2. RELATED WORK

Optimization in optical networks is typically classified as planning/static or online/dynamic. Dynamic optimization refers to making changes while the network operates. Both static and dynamic optimization involve algorithms that range from optimal to heuristics that are typically iterative. They perform intermediate calculations until they find the final solution. At these intermediate calculations, they generally rely on PLMs to take into account the physical layer. The PLMs serve as estimators; they estimate the QoT of unestablished or reconfigured connections [7,8]. The PLM is a model that has several input parameters, which are known with certain accuracy, and thus needs to use appropriate margins for the optimization task at hand [3]. For example, for establishing connections, margins are generally used to model the inaccuracy of the PLM and also to account for the evolution of the physical layer over the lifetime of the connections, increased inference of upcoming connections, equipment aging, etc. Recently ML has been used

to improve the accuracy of the PLM by implementing it with ML models [5] or fitting the parameters of the existing PLM so that its estimations match those monitored in the network [6,8].

The optimization problems in optical networks are multidimensional and combinatorial; a change in one variable affects several others. For example, an establishment of a new connection or a change in a single transponder launch power results in variations in the QoT of all interfered connections. The effect of a few reconfigurations is relatively easy to predict. However, in complex/multivariable optimization problems, if the algorithm at intermediate steps has assumed several reconfigurations, it can project the network into states where the PLM suffers from low accuracy. This would mislead the subsequent calculations and result in poor optimization.

Regarding launch power optimization, several works have appeared aiming at the minimization of nonlinear self- (or intrachannel) and more importantly the cross- (or interchannel) interference effects [2,26–30]. These cross-channel nonlinearities (XCIs) create interdependencies among the launch powers of the connections that share a link, making the problem more complex, as mentioned in the previous paragraph. The authors of [2,26] presented several approaches targeting the optimization of the launch powers of all the channels before establishment (static problem), with the objective of maximizing the network spectral efficiency. Specifically, [26] discussed the potential network level gains achieved by optimizing the power, constellation, and route and wavelength allocations, considering the Gaussian noise (GN) model [27] as the PLM. The parameters of the PLM, such as fiber nonlinearity, attenuation, dispersion coefficients, transponder mismatch loss, and amplifier (flat) gain, were assumed to be fixed during the optimization task. Also, other previous works were based on a fixed parameter PLM and proposed heuristics to optimize all channels' launch powers [1,28]. Note that a fixed parameters PLM works fine for a few reconfigurations, but when the algorithm decides on extensive reconfigurations, and in particular, the adjustment of the launch powers of several connections, the accuracy of the PLM can become critical. The above works select a different launch power for each connection, assumed to be set at each span that the connection crosses. The local optimization leads to the global optimization (LOGO) method [27] and maximizes each span's SNR, assuming the same power for all connections crossing it. Drawbacks are that LOGO assumed spans with a full load and cannot transfer margins among the connections.

The authors of [29] formulated the problem of optimizing the launch powers of all connections to maximize the sum or the minimum channel margin using a PLM based on the GN model (with fixed parameters) as a convex optimization problem. An extension of [29] that improves the SNR estimation accuracy from measurements (thus assuming an operating network/dynamic optimization) was presented in [30]. The authors proposed to probe (change the launch power) and monitor the network, and use that to calculate the partial derivatives needed by the convex optimization algorithm's intermediate calculations. The limiting factors of that work were the assumption of perfect nonlinear impairment monitoring, which is generally considered very hard, along with the extensive interactions with the network for probing. Additionally, the analysis was focused on a single link.

Similar PLM accuracy issues arise in other multivariable dynamic optimization problems, such as dynamic resource allocation, automatic network reconfiguration, defragmentation, and virtual network reconfiguration [9,16,21–25], where the optimization algorithm relies on the PLM to perform calculations for candidate reconfigurations. The PLM in those related works was assumed to have fixed parameters or was aligned before the optimization task and was used to make decisions that were afterward configured to the network. For the extensive reconfigurations targeted in the above works, the PLM can fail because its accuracy drops as the algorithm in its intermediate calculations projects the network into new states.

We here propose to use an iterative closed control loop to solve dynamic multivariable optimization problems. A key part of the proposed iterative closed control loop is that after a number of optimization algorithm intermediate calculations, we close the loop, configure the network, and realign the PLM with the real world via monitoring and ML. By introducing these retraining cycles, the PLM represents the real physical system with enough accuracy to perform the optimization task at hand. The PLM becomes a DT, a model of the system with parameters that evolve/adjust, and a means of dynamically adjusting it.

Closed control loops have been extensively studied in control theory as discussed in [11,31]. However, control theory typically targets infinite time horizon problems, and considers fast loops with real-time feedback. Also, reinforcement learning has received attention on similar topics [32]. Reinforcement learning also targets infinite time horizon problems and a system described by a Markov decision process, which is different from the convex optimization problem that we have at hand.

To the best of our knowledge, the identified issue of the lack of accuracy of the PLM in dynamic optimization problems has not been studied in the past. Note that convex optimization algorithms and their interaction with a tool that represents reality (in optimization terms this is referred to as an "oracle") have been studied [33], including exact and inexact oracles with varying inaccuracy models. Our proposed solution shares certain ideas from this optimization field. We use convex algorithms to solve the launch power optimization problem, following [29,30], but we avoid heavy monitoring and apply the optimization to the network level. We also share ideas with [7,8,12] on the use of monitoring and ML to train/align the PLM with the physical layer conditions. However, we extend those and retrain/realign the PLM in a closed control loop, targeting dynamic optimization problems.

Finally, we would also like to note that our study is quite a bit more realistic than most previous works that use the same PLM as both the estimator and the ground truth to generate the information to train the estimator. In particular, in our simulations, we used VPI as the ground truth and the GN model as the estimator. VPI is more detailed and complex and closer to a real system than the GN model. This choice was made to capture the mismatch between the real network and the PLM that would be used in the optimization process, an additional difficulty that has been neglected in most previous works.

3. USE CASE AND MOTIVATION

In this paper, we investigate how PLM accuracy affects optimization calculations. To do this, we focus on a dynamic version of the launch power optimization problem. We assume that a set of connections are established, and our goal is to optimize their launch power. Thus, no connections are established or released, but the existing connections are reconfigured (their launch powers are adjusted) as the network operates. To motivate and better understand the problem in this section, we discuss the optimization of the launch powers of 25 channels transmitted over a single link.

A. PLM Training

We created a single link with six identical spans set up in VPItransmissionMaker [34] as shown in Fig. 1. On this link we simulated the transmission of 25 channels at 32 Gbaud with polarization multiplexed 16-ary quadrature amplitude modulation (PM-16QAM) and an assumed SNR threshold of $\mathrm{SNR}_{\mathrm{th}} = 13.9$ dB for each [1,6]. Note that these simulations were time-consuming due to the high computational complexity, as VPI uses split-step Fourier propagation simulations to model the nonlinear signal propagation of the channels. We considered the VPI setup as the "real world," the actual optical network.

We also implemented a PLM, and in particular the GN model [27], with a similar setup of six identical spans and 25 channels. We found approximately 1 dB of maximum SNR difference between the PLM and VPI, when all parameters of the PLM and VPI (dispersion coefficient, slope, attenuation coefficient of fiber, nonlinearity coefficient, etc.) were set to be equal. Then we aligned the PLM with the real world (VPI). This alignment can be done with various methods. In our case, we monitored the channels' SNR values (in VPI) and adjusted the PLM parameters so that its SNR estimations match with the real world (VPI), using ML.

To be more specific, we implemented the following alignment process for the GN model [5,6]. We assume a network with N connections. The GN PLM is a model that takes as input several parameters and calculates the SNR values of the connections. Let $\boldsymbol{r}$ denote the set of GN model fitted parameters: i) fiber attenuation coefficients; ii) fiber nonlinear coefficients; iii) fiber dispersion coefficients; iv) a wavelength-dependent penalty term, implemented as a fourth-order polynomial to cover transponder loss mismatches, amplifier ripple, etc.; and v) a bias. Also let $\boldsymbol{p} = [p_1, p_2, \ldots, p_N]$ be

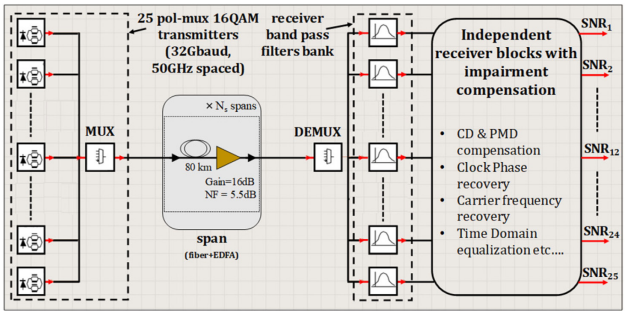

Fig. 1. Simulated single-link setup in VPI with 25 × 32 Gbaud, PM-16QAM, 50 GHz spaced transmitters, and six identical spans.

the launch power vector of the N connections, which are the variables that will be optimized later, and let $\boldsymbol{z}$ represent the unchanged input parameters for our optimization, such as routes, used wavelengths, and span lengths. We denote by $Q_n(\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{z})$ the GN model SNR estimation for connection n, and with $\boldsymbol{Q_N}(\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{z})$, the SNR vector for all connections N. The SNR calculation function is nonlinear in its parameters $\boldsymbol{r}$ (and also $\boldsymbol{p}$). Finally, let $Y_n(\boldsymbol{p})$ denote the monitored SNR value of connection n and $\boldsymbol{Y_N}(\boldsymbol{p})$ denote the vector for all the connections N. In this work, such monitoring is assumed to be done at the coherent receivers. The training error vector is given by $\boldsymbol{Q_N}(\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{z}) - \boldsymbol{Y_N}(\boldsymbol{p})$, and the objective of the fitting is to identify the parameters $\boldsymbol{r}$ that minimize the squared error. To fit this, we relied on the Levenberg–Marquardt (LM) algorithm, which is suitable for solving nonlinear least squares fitting problems [35]. The LM algorithm finds

$$\boldsymbol{r}_0 = \mathrm{argmin}_{\boldsymbol{r}} \left(\boldsymbol{Q_N}(\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{z}) - \boldsymbol{Y_N}(\boldsymbol{p}) \right)^2. \tag{1}$$

When we perform this PLM alignment once, before the optimization task, the PLM reflects with good accuracy the starting state of the network prior to optimization. We refer to this as one-time trained PLM and denote it by $\boldsymbol{Q_N}(\boldsymbol{p}, \boldsymbol{r}_0, \boldsymbol{z})$.

We studied two types of erbium-doped fiber amplifiers (EDFAs): one whose gain is perfectly flat/ideal and another with a gain ripple profile of a peak-to-peak (p2p) value of ±0.2 dB [6]. Note that the EDFAs were assumed to be operated in automatic gain control (AGC) mode, with average gain equal to the previous span loss. We call the setup with the flat span EDFAs as Case 1, and the setup with EDFAs having gain ripple as Case 2. Case 1 represents an ideal network with relatively stable physical layer conditions. On the other hand, Case 2, with rippled EDFAs, represents a more realistic scenario with more volatile/dynamic physical layer conditions. The physical layer dynnamics comes from the fact that an EDFA with a gain ripple introduces SNR variations when changing the connection powers. These variations are hard to estimate unless we exactly know the gain profile of the EDFA. This profile is hard to be found in an operating network, and it might change over a long time.

Figures 2(a) and 2(b) show the estimated SNR values of the connections from the one-time trained PLM $\boldsymbol{Q_N}(\boldsymbol{p}, \boldsymbol{r}_0, \boldsymbol{z})$ and the real network (VPI) $\boldsymbol{Y_N}(\boldsymbol{p})$ at uniform launch power of $\boldsymbol{p} = 0$ dBm, for flat and rippled EDFAs, respectively. The corresponding training errors are also displayed in the same figures. With one-time training, the PLM parameters were adjusted quite well, and its estimated SNR values matched those of the actual optical network/real world at the initial state. This is deduced by the very low errors, less than 0.1 dB for both Case 1 and Case 2.

B. Dynamic Launch Power Optimization

We now turn our attention to the dynamic launch power optimization problem. For a generic topology, we assume that we have a set of N established connections. The objective is to optimize the launch powers of the N transponders to maximize

(i) objective 1: sum of connections margins,

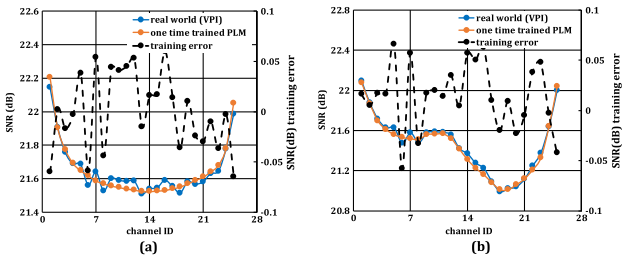

Fig. 2. Estimated SNR values from the one-time trained PLM and VPI at uniform 0 dBm launch power and the related training error for (a) a flat EDFA and (b) an EDFA with gain ripple.

$$\max f(\boldsymbol{p}) = \sum_{n=1}^{N} \left(\log \mathrm{SNR}_n(\boldsymbol{p}) - \log \mathrm{SNR}_{\mathrm{th},n}\right);$$

(ii) objective 2: minimum margin,

$$\max f(\boldsymbol{p}) = \min_{n\in[1,N]} \left(\log \mathrm{SNR}_n(\boldsymbol{p}) - \log \mathrm{SNR}_{\mathrm{th},n}\right)$$

subject to

$$\log \mathrm{SNR}_n(\boldsymbol{p}) - \log \mathrm{SNR}_{\mathrm{th},n} \geq 0, \quad \forall n \in [1, N],$$

$$\boldsymbol{p}_{\min} \leq \boldsymbol{p} \leq \boldsymbol{p}_{\max},$$

where $\boldsymbol{p} = [p_1, p_2, \ldots, p_N]$ is the launch power vector of the N connections, $p_{\min}$ and $p_{\max}$ are the lower and upper power limits of the transponders' launch power, $\mathrm{SNR}_n(\boldsymbol{p})$ is the SNR of connection n for the corresponding power vector $\boldsymbol{p}$, and $\mathrm{SNR}_{\mathrm{th},n}$ is the SNR threshold required for the modulation format of n.

The optimization of the channels' launch powers with one of the above objectives is known to be convex and of polynomial complexity [29,30]. Hence, we used the interior-point algorithm to solve it. In general, a convex optimization algorithm performs intermediate calculations/iterations. At each intermediate iteration, the algorithm decides on new transponder launch powers to move towards the optimum. However, these intermediate steps are internal; only the final (optimal) will be configured in the network. The algorithm decides these steps by using the knowledge of the partial derivatives (first and sometimes second order, depending on the algorithm) of the objective and constraints with respect to the variables (launch powers $\boldsymbol{p}$).

The first option to calculate such derivatives is to interface the optimization algorithm with a PLM. In this case, the $\mathrm{SNR}_n(\boldsymbol{p})$ values in the algorithm come from the PLM calculations $Q_n(\boldsymbol{p}, \boldsymbol{r}, \boldsymbol{z})$. If the PLM has closed-form partial derivatives, then the optimization process is straightforward. However, typically the PLMs (e.g., the GN model) do not have closed-form derivatives. Then we can use a derivative identification subroutine based on finite differences. This subroutine makes changes in the launch powers and uses the PLM to calculate the outcomes (connections' new SNR values). In this section, we will assume that we use a PLM that was aligned/trained once before the optimization, as discussed above, so we use the fitted parameters $\boldsymbol{r} = \boldsymbol{r}_0$. We will refer to this as optimization with a one-time trained PLM.

An alternative option is to probe the real network, that is, to interface the algorithm and, in particular, the derivative identification subroutine, with the network, bypassing the PLM. In this case the $\mathrm{SNR}_n(\boldsymbol{p})$ values in the algorithm come from the monitors of the network $Y_n(\boldsymbol{p})$. The derivative identification process would configure through the control plane the launch powers of the transponders and would monitor the outcomes (connections' SNRs) to calculate the derivatives. This would be repeated at each algorithm's iteration. We will refer to this option as optimization with monitoring probes.

Note that the former option is fast. The PLM is trained once and used thereafter to compute the derivatives. Although the PLM is called several times, it has low computation complexity (at least the GN model), resulting in low overall optimization time. However, this option suffers from accuracy issues. Specifically, several parameters such as the amplifier gain ripple, nonlinear interference (NLI), cross talk at switches, etc., change for different network configurations/states. So, the one-time trained PLM, which is quite accurate at the beginning/initial state [Figs. 2(a) and 2(b)] behaves rather inaccurately as the iterative optimization algorithm projects the network into states that are away from the initial. The inaccuracy of the PLM results in inaccurate estimation of the derivatives, which in turn results in suboptimal optimization of the launch powers. This accuracy problem is expected to be more profound when the network physical layer is more dynamic, as in Case 2, where EDFA gain ripples result in SNR variations as the launch powers change.

On the other hand, the latter option, optimization with monitoring probes, involves several interactions with the actual network at each intermediate step, which typically take a long time and are also susceptible to monitoring errors. Note that the term monitoring probes refers to the capability of the network to change the launch powers and monitor the outcomes. In other optimization problems, e.g., involving establishment/release of connections, such capability will probably not be present. Finally, note that in the results presented here and in Section 5 up to Fig. 12, the monitoring error was assumed to be zero. Thus, the results obtained with monitoring probes and zero monitoring error are optimal and set as the reference for all other cases.

C. Deviation of Optimizing with the One-Time Trained PLM

Figures 3(a) and 3(b) show the optimized launch powers for obj#1 with the one-time trained PLM and the monitoring probes approaches for flat EDFAs (Case 1). We see that the one-time trained PLM did not support the optimization well, since the algorithm using it identified quite different power levels. That is, although the algorithm using the one-time trained PLM identified the optimum, this was optimum for the PLM and not close to the optimum in the real network (VPI). The reason for this is that the PLM could not follow/predict with good accuracy the real SNR values at the power levels calculated by the algorithm, although the accuracy was very good for the initial state of the network, right after the (one-time) training. A margin on the PLM could cover this, but again would result in suboptimal calculations. The maximized sum

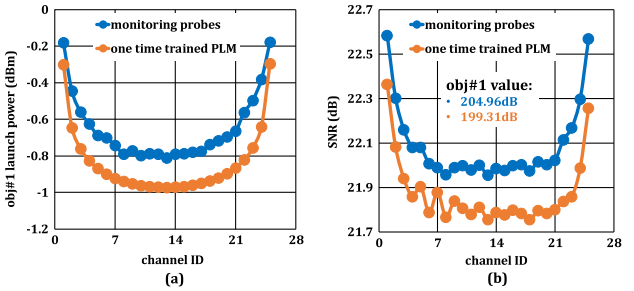

Fig. 3. (a) Optimized launch powers and (b) corresponding SNR and obj#1 value, evaluated in the real network (VPI), for Case 1 (flat EDFAs).

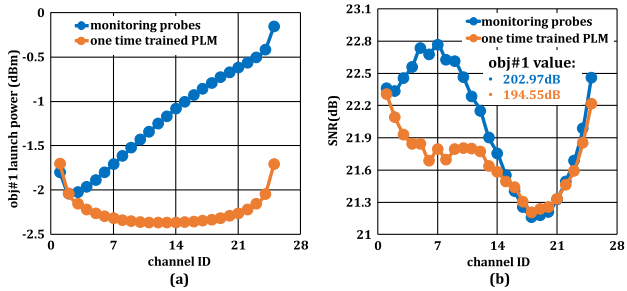

Fig. 4. (a) Optimized launch power and (b) corresponding SNR and obj#1 value, evaluated in the real network (VPI), for Case 2 (EDFA with gain ripple of ±0.2 dB).

of SNR margins (obj#1) was optimized to 204.96 dB when the algorithm used monitoring probes and interacted with the real network, and to 199.31 dB when it interacted with the one-time trained PLM. There exists a mismatch of ∼5.6 dB in the obj#1 value between these two optimization approaches. Note that the SNR values and the objective [Fig. 3(b)] were and should be evaluated in the real world (VPI), so that we can see the deviation. This also explains the ripples in SNR seen in Fig. 3(b), since in VPI models some wavelength-dependent factors are not covered by GN. Similar behavior was observed for obj#2, not shown here for conciseness. Note that optimization with obj#2 results in choosing the launch powers that result in almost flat SNR values, since maximizing the minimum margin iteratively pushes the lowest SNR value and reduces the higher. The maximized minimum margin was optimized to 7.64 dB with monitoring probes, and to 7.36 dB with the one-time trained PLM.

To emulate a more realistic scenario, we assigned a gain ripple profile to all EDFAs having a p2p ripple value of ∼ ±0.2 dB (Case 2). In such a scenario, when the algorithm used the one-time trained PLM, it reached an optimization objective (evaluated in the real network—VPI) quite worse than when it used monitoring probes and interacted with the actual network at intermediate optimization iterations. Figures 4(a) and 4(b) show the optimized launch powers and their corresponding SNR values, respectively, for obj#1. A maximum input power difference of ∼1.5 dBm was observed, resulting in ∼8.4 dB of SNR difference for obj#1. Similarly, for obj#2, we observed a maximum input power difference of ∼1.2 dBm, resulting in ∼0.62 dB of SNR difference for obj#2. Note that the mismatch is higher than previously (Case 1/flat EDFAs). This is because we now have a more volatile physical layer (EDFA gain ripples affect the SNRs), and the PLM we use does not cover this additional volatility.

Concluding, for any optimization problem, the PLM accuracy is important. For planning/static problems, we cover the inaccuracy issue with margins, while several papers have targeted the reductions of margins by aligning the PLM to the physical layer conditions, e.g., through monitoring and ML. However, there have been limited discussions on dynamic optimization problems; the disadvantage is that to justify dynamic optimization, we should target to achieve high efficiency, making the accuracy of the PLM more critical. For complex/multivariable dynamic optimization tasks, such as the dynamic launch power optimization problem discussed above, an iterative algorithm is typically used that calculates several intermediate solutions. One option is to interface the algorithm with the network to probe and monitor it in order to carry out the intermediate steps until it achieves the optimum. This, however, is cumbersome and very slow. On the other hand, we can train the PLM before the optimization and use it in all intermediate calculations. Since PLM calculations are fast, the optimization will finish quickly. However, the accuracy of the PLM can deteriorate and result in suboptimal optimization, as seen in the preliminary results discussed above. This motivated us to address the limitations of the optimization with a one-time trained PLM by exploring the operating network and its feedback. Our goal is to appropriately realign the PLM at intermediate optimization calculations so that the difference between the optimization objective achieved with monitoring probes (interacting with the real world) and with the retrained PLM is negligible, while the whole optimization is much faster.

4. NETWORK DYNAMIC OPTIMIZATION AND PLM RETRAINING

PLMs, which can be analytical, semianalytical, ML models, etc., have certain accuracy. The modeling assumptions impact the estimation accuracy. For example, many PLMs neglect EDFA gain ripples, partially model NLIs (e.g., consider full load), filters' [inside reconfigurable optical add-drop multiplexers (ROADMs)] cross talk, residual dispersion, and specific parameters of transponders. Note that a detailed PLM is slower in the calculations and requires more input parameters. Then, a second factor comes into play: the input parameters might not be known with good accuracy, which eventually reduces the accuracy of a detailed PLM.

Optimization tasks result in network changes and typically use a PLM to estimate the effect of such changes. However, these changes also modify the physical layer itself; they move the network to a new state. Depending on the PLM, such changes are covered to a certain degree. For example, when changing the power of a connection, NLIs and cross talk, but also the penalties due to EDFAs' gain ripple profiles, change. The PLM model could, for example, cover the effect of NLIs and cross talk, but not the evolution of gain ripples. For these reasons, margins are used. However, in dynamic use cases, the aim is to be more efficient, and thus the accuracy of the PLM is

a crucial factor. To improve that, we can take advantage of the operating network.

Hence, a basic need for dynamic optimization is to have a PLM that follows the network changes. In the AI/ML era, a way to do this is to choose an appropriate set of parameters and retrain the PLM at certain points. However, retraining is cumbersome, and thus we cannot retrain it before every dynamic task. On the other hand, training the PLM once before a multivariable optimization task can result in suboptimal optimization, since the accuracy of the PLM deteriorates after several intermediate calculations.

In this paper, we focus on dynamic multivariable optimization problems, and, in particular, we study the launch power optimization problem of established connections as introduced in the previous section. Note, however, that the proposed solution is generic and applicable to other dynamic simple or multivariable optimization problems as well. We propose to use an iterative closed control loop process to solve such dynamic multivariable optimization problems. At certain intermediate iterations of the algorithm, we close the loop, configure the network, and monitor to retrain the PLM (with ML) to follow the projected network conditions. The target is to make the PLM a digital replica, that is, a DT, of the optical physical layer for the dynamic optimization task at hand. The frequency of the PLM retraining depends on the PLM model and on the optimization task. As discussed in the previous section, alternative options for the algorithm are to avoid using a PLM and have the algorithm interact with (probe and monitor) the network or use a one-time trained PLM. All three options are formally described in the following subsections.

A. Optimization with Monitoring Probes

The scheme that is considered in this subsection assumes that the optimization algorithm interacts directly with the actual network and follows a closed control loop process. The algorithm employs a subroutine to specify the probes, the configurations that are applied to the network. Then it monitors the outcomes to identify the information that it needs for an intermediate optimization step. A representation of this scheme is shown in Fig. 5(a).

To be more specific, we focus on the dynamic launch power optimization problem with a typical objective, such as maximizing the sum of SNR margins, or the minimum margin, as discussed in Section 3. This problem is known to be convex and of polynomial complexity. The convex optimization algorithms, such as (sub)gradient methods, interior point, trust-region-reflective, etc., are iterative; at each iteration, they need to calculate Jacobians and/or Hessians for the objective and constraint functions [36,37]. Actually, the related algorithms are classified into first or second order depending on the order of the partial derivatives they use. For optimization problems that involve PLMs without closed-form partial derivatives, a way to calculate them is to use a subroutine that implements finite differences [37].

For example, for the power optimization problem at hand, to calculate the gradient for an objective function f we need to find/monitor the changes in the SNR values of all connections, assumed to be done through the coherent receivers, with respect to changes in the powers of the transponders. To give an example, assume a network with a set of N established connections with a launch power vector $\boldsymbol{p}$. We denote by $\boldsymbol{\delta p_n}$ the vector with all zeros apart from element n whose value we set to p_{step}, what we refer to as the power probe step. As a matter of fact, the change in launch power of the single connection n results in changes in the SNR values of all interfered connections (those that share a common link). So, we denote by $\boldsymbol{SNR_N(p)}$ and $\boldsymbol{SNR_N(p+\delta p_n)}$ the SNR vector of all N connections for the respective power vectors. If f is the objective function, then the first-order partial derivative for n is given by

$$g_n = f(\boldsymbol{p}) - f(\boldsymbol{p}+\boldsymbol{\delta p_n})/p_{\text{step}}. \tag{2}$$

Depending upon the function f, this involves certain operations with the vectors $\boldsymbol{SNR_N(p)}$ and $\boldsymbol{SNR_N(p+\delta p_n)}$. The gradient g is the vector of all partial derivatives, that is, g_n for all n. To calculate the gradient with the finite-difference method, we need to probe with $\boldsymbol{p}+\boldsymbol{\delta p_n}$ and

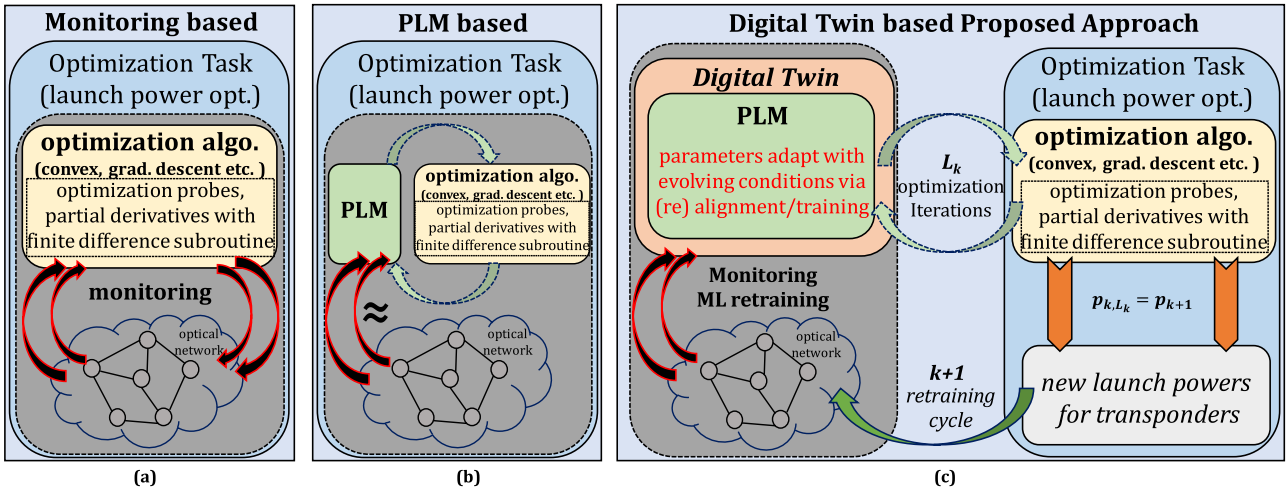

Fig. 5. Optimization with (a) actual deployed monitors, (b) a one-time trained PLM, and (c) the proposed PLM retraining DT approach.

Pseudo-code - 1

Start with initial launch power vector $\boldsymbol{p_0}$, iteration number i=-1
While not converged
  Increase i
  Finite differences process, time: t_{mon} per probe to monitor all connections $\boldsymbol{Y_N}$
    Probe d_i times (configure new launch powers and monitor) (e.g. for the first order partial derivative of connection n, probe with $\boldsymbol{p_i} + \boldsymbol{\delta p_n}$ and monitor $\boldsymbol{Y_N}(\boldsymbol{p_i} + \boldsymbol{\delta p_n})$)
  Calculate the derivatives and next launch power vector $\boldsymbol{p}_i$, time: t_{calc}
  Evaluate convergence (e.g. compare objective improvement with a threshold), when converged: $L_{mon\_prob} = i$
  Apply the calculated power vector p_i to the network

Fig. 6. Pseudo-code for optimization with monitoring probes.

monitor $\boldsymbol{SNR_N}(\boldsymbol{p} + \boldsymbol{\delta p_n})$, and repeat this probe/monitoring process for all connections $n = 1, \ldots, N$.

Generalizing this, the optimization algorithm calculates (first- or second-order) partial derivatives through a finite-differences subroutine at each intermediate iteration. Let us assume that the algorithm calls the finite-differences method d_i times at iteration i. For the above example with the gradient, we have $d_i = N$. This is the simplest case; we typically have $d_i \geq N$, depending on the algorithm. Note also that we might have different numbers of probes per iteration, that is, different d_i per i. However, to simplify our analysis, we assume that this is constant, $d_i = D$, for each iteration i.

The convex optimization algorithm with monitoring probes performs $L_{\text{mon\_prob}}$ iterations to find the optimum. We also denote by t_{mon} the monitoring time, assumed here to monitor simultaneously all N-established connections. The monitoring time can range from minutes to hours, depending upon the network size, the monitoring plane, the targeted monitoring error, etc. [38]. However, once the monitoring information is forwarded to the algorithm, the time t_{calc} to calculate the gradients/Hessian and also the next launch powers is quite lower (millisecond range) compared to the monitoring time ($t_{\text{mon}} \gg t_{\text{calc}}$). So, with the monitoring probes-based approach, under the assumption that N connections are monitored in parallel, the total optimization time $T_{\text{mon\_prob}}$ is given by

$$T_{\text{mon\_prob}} = L_{\text{mon\_prob}} \cdot (D \cdot t_{\text{mon}} + t_{\text{calc}}) \approx L_{\text{mon\_prob}} \cdot D \cdot t_{\text{mon}}. \quad \textbf{(3)}$$

Optimizing with monitoring probes is described with pseudo-code 1 in Fig. 6.

In general, the optical monitors have certain measuring accuracy. In our proposal, we assume that we monitor the SNR from the coherent receivers, which are quite accurate. Note that higher accuracy can be achieved through time averaging; to reduce the effect of short-term time impairments, e.g., polarization, the monitoring measurements could be averaged over time, resulting in higher accuracy but also higher monitoring time. Depending on the monitoring error, we might end up with a different and worse objective value instead of the optimum. Another factor to be accounted for in a real network is that the power probe steps (p_{step}) cannot be very small because fine-tuning of the equipment is not feasible. Thus, in a real network, there are two factors that hinder the monitoring probe optimization process:

(i) the monitoring errors,
(ii) the minimum power probe step p_{step} that can be configured.

We call the SNR vector obtained from monitors with errors as the noisy monitored vector, and denote it by

$$\tilde{\boldsymbol{Y}}_N(\boldsymbol{p}) = \boldsymbol{Y}_N(\boldsymbol{p}) + \boldsymbol{v}, \quad \textbf{(4)}$$

where $\boldsymbol{v}$ is a vector that represents the monitoring error (or noise).

Stochastic subgradient methods [39] for zero mean errors provably find the optimum solution with specific step sizes but might require a very large number of iterations. However, in a real network where the use of small steps is not supported by the transponders and iterations are expensive since they involve several monitoring phases, such methods are hardly applicable.

In this monitoring probes optimization approach, the algorithm optimizes the launch powers and checks the actual conditions of the network at each step. For zero error, this approach identifies the optimum $\text{obj}_{\text{mon\_prob}}$ but requires a high optimization time $T_{\text{mon\_prob}}$. So, we will use this as the reference for all other approaches.

Also note that the use of monitoring probes, which are used in this method to identify the partial derivatives, is not a universal solution. A monitoring probe in the studied use case refers to the configuration of new launch power(s) to one (or more) transponders of the established connections and monitoring of all connections' SNRs at their receivers. So, the definition is specific to the problem; different optimization problems require different monitoring probe definitions. For some tasks, monitoring probes might not be available, e.g., tasks involving the establishment/release of connections. For such tasks, we might need spare transponders to extract the information required for the optimization [12], which implies higher cost and complexity.

B. Optimization with One-Time Trained PLM

In this subsection, we consider the method where the PLM is aligned only once, at the beginning of optimization. We perform the alignment of the PLM using monitoring information $\boldsymbol{Y}_N(\boldsymbol{p}_0)$ from the actual network, assumed to take time t_{mon}, as above. We then use ML to fit the parameters $\boldsymbol{r}$ of the PLM $\boldsymbol{Q}_N(\boldsymbol{p}_0, \boldsymbol{r}, \boldsymbol{z})$ to the physical layer conditions, so as to identify $\boldsymbol{r}_0$. This is assumed to take time t_{train}. Then the optimization algorithm interacts with this one-time trained PLM, $\boldsymbol{Q}_N(\boldsymbol{p}_0, \boldsymbol{r}_0, \boldsymbol{z})$, at each intermediate step, to estimate the QoT (SNR) of the connections as shown in Fig. 5(b). In particular, since there are no closed-form derivative equations for the GN model, we use a similar derivative identification subroutine (finite differences), as in the previous method, but this time we interface that with the PLM instead of the actual network. We denote by t_{PLM} the time that the PLM takes to calculate the SNR values of all connections. As before, this subroutine is assumed to be called D times at each algorithm intermediate iteration. We assume that the time t_{calc} that the algorithm needs to calculate the gradients/Hessian and also the next launch powers is the same as the previous method.

Pseudo-code - 2

Start with initial launch power vector $\boldsymbol{p_0}$, iteration number i=-1
Align PLM to initial / prior-to-optimization state (monitor $\boldsymbol{Y_N(p_0)}$ and train the PLM $\boldsymbol{Q_N(p_0, r, z)}$ to find r_0), time: $t_{mon} + t_{train}$
While not converged
  Increase i
  Finite differences process, time : t_{PLM} per SNR vector calculation by the PLM $\boldsymbol{Q_N}$
    Probe d_i times the PLM: change the launch powers and calculate the SNR vector for all connections with the PLM
    (e.g. for the first order partial derivative of connection n, set $\boldsymbol{p_i + \delta p_n}$ and calculate $\boldsymbol{Q_N(p_i + \delta p_n, r_0, z)}$),
  Calculate the derivatives and next launch power vector $\boldsymbol{p_i}$, time: t_{calc}
  Evaluate convergence (e.g. compare objective improvement with a threshold), when converged: $L_{PLM} = i$
Apply the last calculated power vector $\boldsymbol{p_i}$ $(= \boldsymbol{p_{L_{PLM}}})$ to the network

Fig. 7. Pseudo-code for optimization with a one-time trained PLM.

We also denote by L_{PLM} the number of iterations that the algorithm performs. With the one-time trained PLM, the total optimization time T_{PLM} is given by

$$T_{\text{PLM}} = t_{\text{mon}} + t_{\text{train}} + L_{\text{PLM}} \cdot (D \cdot t_{\text{PLM}} + t_{\text{calc}}) . \quad \textbf{(5)}$$

It stands to reason that the PLM training and estimation calculations and the algorithm calculations are substantially faster than monitoring ($t_{\text{mon}} \gg t_{\text{train}}, t_{\text{PLM}}, t_{\text{calc}}$). We also expect a similar number of iterations ($L_{\text{PLM}} \approx L_{\text{mon\_prob}}$), because the PLM/GN model satisfies the convexity properties [29]. So, we have $T_{\text{PLM}} \approx t_{\text{mon}}$. By comparing this to Eq. (3), we can see that the one-time trained PLM-based optimization approach requires substantially less optimization time than the previous approach, $T_{\text{PLM}} \approx t_{\text{mon}} \ll T_{\text{mon\_prob}} \approx L_{\text{mon\_prob}} \cdot D \cdot t_{\text{mon}}$. In particular, the speedup we obtain is of the order of $L_{\text{mon\_prob}} \cdot D$. This happens because the one-time trained PLM quickly provides all the necessary information for the optimization algorithm at each intermediate step, avoiding monitoring. Optimizing with a one-time trained PLM is described with pseudo-code 2 in Fig. 7.

The optimization algorithm using the one-time trained PLM identifies the launch powers that yield the optimum $\tilde{\text{obj}}_{\text{PLM}}$, but this is viewed through the PLM. However, the PLM has certain accuracy and was trained at initial conditions. So the identified launch powers yield the objective obj_{PLM} in the real network, which is worse than the objective of the monitoring probe method, which is always evaluated in the real network, $\text{obj}_{\text{PLM}} \leq \text{obj}_{\text{mon\_prob}}$. This problem was identified in Section 3 and shown in Figs. 3 and 4.

C. Optimization with a DT

Following the discussions of the two above approaches, and the results presented in Section 3, we observe a clear trade-off between optimization time and performance. The monitoring probes-based approach [in Fig. 5(a)] implements closed control loops that are not fast, due to the complex probing and slow monitoring subroutine. However, it achieves real optimization as it tracks the network evolving conditions/states by configuring and monitoring. On the other hand, the one-time trained PLM approach [in Fig. 5(b)] is substantially faster because it uses the PLM to quickly find the derivatives at intermediate states. However, the PLM is trained only once, at the beginning of the optimization. So, if the algorithm projects the network to substantially different physical conditions, then the optimization is suboptimal, since the PLM differs from reality, as seen in Figs. 3 and 4. The following proposed scheme keeps the benefits of both approaches: it finds a near-to-optimal solution, but with an overall low optimization time.

We propose to use an iterative closed control loop process to solve the dynamic optimization problem. At certain intermediate iterations of the algorithm, we configure the intermediate solution to the network and monitor to realign the PLM (with ML) to follow the projected network conditions, as shown in Fig. 5(c). The idea is to make the PLM a DT, to have a PLM model that is parametric, and to define the method to readjust/realign it to represent the physical system with enough accuracy to perform the dynamic optimization calculations at hand. For realigning the PLM, many techniques can be used. We here use ML training. In this study, we used as the PLM the GN model [27], which considers the launch powers and wavelength occupancy. Thus, it models quite accurately linear and NLI transmission impairments. We also extended it and added a wavelength-dependent penalty on top of the GN SNR calculation to cover, e.g., the EDFA ripple penalties [6]. The GN alignment process was described in Section 3, and extended here to be performed iteratively.

As above, we denote by $\boldsymbol{Q_N(p, r, z)}$ the calculation of the SNR values vector of all N connections by the GN PLM, where $\boldsymbol{p}$ is the launch power vector (optimization variables), $\boldsymbol{r}$ represents the PLM fitted parameters, the fiber coefficients, and the wavelength-dependent ripple penalty, and $\boldsymbol{z}$ represents the unchanged input parameters for our optimization such as routes and wavelengths used. The dynamic optimization process starts with the configured launch power vector $\boldsymbol{p_o}$ of the established connections (e.g., all 0 dBm). For this initial power vector $\boldsymbol{p_o}$, the PLM is trained with the monitored SNR vector $\boldsymbol{Y_N(p_o)}$. To be more specific, we use ML and in particular the LM algorithm to find $\boldsymbol{r_0} = \text{argmin}_r(\boldsymbol{Q_N(p, r, z)} - \boldsymbol{Y_N(p)})^2$. Now, let us assume that at the end of the kth PLM training cycle, the optimization algorithm has performed L_k intermediate iterations and identified the launch powers $\boldsymbol{p_k^{L_k}}$. We then start the next cycle $k+1$ by configuring the network with the outcome so $\boldsymbol{p_{k+1}} = \boldsymbol{p_k^{L_k}}$. To retrain the PLM for the $k+1$ cycle, we configure the network with $\boldsymbol{p_{k+1}}$ and monitor to obtain $\boldsymbol{Y_N(p_{k+1})}$. Then ML is used to fit the parameters $\boldsymbol{r_{k+1}}$, that is, $\boldsymbol{r_{k+1}} = \text{argmin}_r(\boldsymbol{Q_N(p_{k+1}, r, z)} - \boldsymbol{Y_N(p_{k+1})})^2$. This PLM is then used in the optimization algorithm iterations of cycle $k+1$. Note that, at each retraining cycle of the PLM, we can make use of the previously monitored SNRs, including thus the history and the network evolution conditions. This tends to improve the PLM accuracy as the algorithm iterates, where the accuracy is more critical.

We assume that in total we retrain the PLM K_{retrain} times. Although we can have different numbers of algorithm iterations per cycle, to simplify our analysis in the following, we assume that the algorithm runs L_{iter} iterations after each PLM retraining. So, $L_k = L_{\text{iter}}$ for all retraining cycles $k = 1, \ldots, K_{\text{retrain}}$. It is easy to visualize the overall concept as

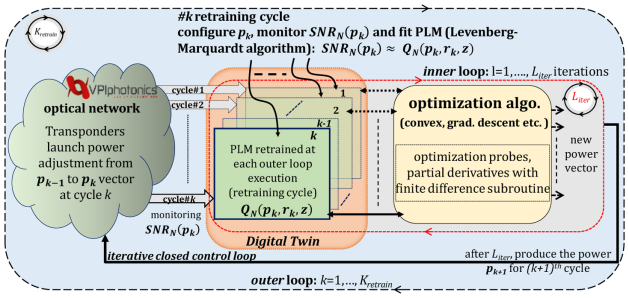

Fig. 8. Schematic showing the two nested "for" loops, the outer for retraining PLM cycles and the inner for the optimization algorithm iterations.

Pseudo-code - 3

Start with initial powers $\boldsymbol{p_0}$, outer loop iteration number k=0
While not converged
  Align the PLM to current network state (monitor $\boldsymbol{Y_N(p_k)}$ and train the PLM $\boldsymbol{Q_N(p_k, r, z)}$ to identify $\boldsymbol{r_k}$, time: $t_{mon} + t_{train}$
  Increase k, $\boldsymbol{p_k^0 = p_{k-1}}$
  For l=0,..., L_{iter}-1(Inner loop iterations)
    Finite differences process, time : t_{PLM} per SNR vector calculation by the PLM $\boldsymbol{Q_N}$
      Probe d_i times the PLM: change the launch powers and calculate the SNR vector of all connections with the PLM (e.g. for the first order partial derivative of connection n, set $\boldsymbol{p_k^l + \delta p_n}$ and calculate $\boldsymbol{Q_N(p_k^l + \delta p_n, r_k, z)}$)
    Calculate the derivatives and next power vector $\boldsymbol{p_k^{l+1}}$, time: t_{calc}
  Evaluate convergence (e.g. compare objective improvement with a threshold), when converged: $K_{retrain} = k$
  Apply the calculated power vector $\boldsymbol{p_k^{L_{iter}}}$ to the network

Fig. 9. Pseudo-code for optimization with the proposed PLM retraining/DT.

two nested "for" loops, as shown in Fig. 8. The outer one pertains to the PLM retraining, and the inner to the optimization algorithm intermediate iterations with the retrained PLM/DT. The time for each retraining cycle is equal to T_{PLM} for L_{iter} iterations, that is, $t_{\text{mon}} + t_{\text{train}} + L_{\text{iter}} \cdot (D \cdot t_{\text{PLM}} + t_{\text{calc}})$. We denote the overall optimization time with this DT-based approach as T_{DT}, which is given by

$$T_{\text{DT}} = K_{\text{retrain}} \cdot (t_{\text{mon}} + t_{\text{train}} + L_{\text{iter}} \cdot (D \cdot t_{\text{PLM}} + t_{\text{calc}})) . \quad \textbf{(6)}$$

The proposed method of optimizing with a DT is described with pseudo-code 3 in Fig. 9.

Note that, in total, the optimization algorithm performs $K_{\text{retrain}} \cdot L_{\text{iter}}$ iterations and retrains the PLM K_{retrain} times. Our target is to choose the retraining period L_{iter} appropriately so that the PLM would follow with good accuracy the physical layer in the algorithm's intermediate calculations. If this is achieved, the PLM estimated objective that is calculated at each iteration and the final one $\tilde{\text{obj}}_{\text{DT}}$ would be very close to the real value in the real network obj_{DT}. Also, the achieved objective would be very close to the optimum, as calculated by the monitoring probes method $\text{obj}_{\text{mon\_prob}}$. So, we would have $\tilde{\text{obj}}_{\text{DT}} \approx \text{obj}_{\text{DT}} \approx \text{obj}_{\text{mon\_prob}}$. Moreover, for an appropriate retraining period, the iterations of the optimization algorithm would be close to those of the monitoring probes, that is, $K_{\text{retrain}} \cdot L_{\text{iter}} \approx L_{\text{mon\_prob}}$. Looking at the total optimization times, and assuming that t_{mon} is the dominant factor, we have $T_{\text{DT}} \approx K_{\text{retrain}} \cdot t_{\text{mon}}$. Thus we obtain a speedup of $L_{\text{mon\_prob}} \cdot D/_{\text{retrain}} = L_{\text{iter}} \cdot D$ with respect to the monitoring probes optimization approach.

5. RESULTS AND DISCUSSION

To quantify the benefits of the devised DT-based power optimization approach, we carried out simulations using both VPItransmissionMaker and MATLAB. The actual network was implemented in VPI, and the PLM (relying on the GN model) and the convex optimization algorithm were developed in MATLAB. Note that VPI is more detailed and complex and closer to a real system than the GN model. This choice was made to capture the mismatch between the real network and the PLM used in the optimization process. This is a considerable improvement in terms of realism compared to many prior studies (listed in Section 2), where authors used the same PLM for both the real network/ground truth and their proposed solution.

To be more specific, we implemented in MATLAB the GN model and the launch power optimization algorithm to maximize: (obj#1), the sum of SNR margins, or (obj#2), the lowest margin, as discussed in Section 3. This optimization problem is known to be convex and of polynomial complexity. Hence, we implemented an interior-point algorithm to solve it. The algorithm was run until it found the optimum (optimality tolerance 10^{-6}). The GN model was interfaced with the optimization algorithm, and both were integrated in an automated system in VPI. For each simulation, VPI implements the outer loop (PLM retraining cycle). It takes as input the launch powers coming from the algorithm of the integrated MATLAB module, performs the detailed transmission simulations, and calculates the SNRs of the channels. These are passed as input to the integrated MATLAB module. With that input, the integrated PLM gets trained, and this trained PLM is then used by the convex algorithm for L_{iter} intermediate iterations. In those iterations, the algorithm uses the PLM to identify the partial derivatives, using the finite-differences subroutine, and then the new launch powers. After the L_{iter} iterations (inner cycle), a new set of launch powers are automatically fed to VPI transponders as a closed control loop for the next retraining cycle. Note that, at each retraining cycle of the PLM, we retrained with the current and previously monitored SNRs, including thus the history and the network evolution conditions.

The monitoring probes approach [Fig. 5(a)] was used as a reference in this work. To implement this, we implemented another (more frequent) closed control loop without using a PLM: the monitoring probes from the finite-differences subroutine (in MATLAB) were carried directly to VPI, and the SNR values were then passed back to that subroutine. The one-time trained PLM approach [Fig. 5(b)] represents the traditional optimization scheme via a PLM. For that, we train the PLM only once with the SNR data from VPI at the beginning of the simulation and used that PLM for the power

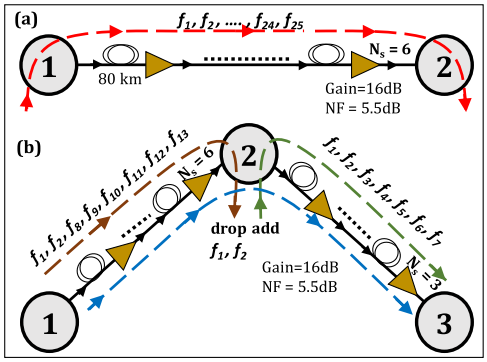

Fig. 10. VPI setup with (a) a single link of six identical spans and (b) three nodes, 15 connections with different paths/routes, and added/dropped points to emulate a small network.

optimization (involving intermediate iterations not configured in the network). Note that in all cases, we start by assigning 0 dBm uniform power to all transponders in VPI.

For all the optimization schemes, the objective value was calculated in VPI, as in the real network. As discussed, there exists a difference between the view of the PLM/optimization process that uses it and the real objective. Finally, note that we evaluated the benefits of our proposed scheme for relatively small channel count (=25) and up to two links, due to the slow execution time of VPI (split-step Fourier simulations). Actually, this can be considered as an indication of the long time of interacting with/monitoring the real network.

To be specific, we made two fully automated setups in VPI:

(i) a single link of six identical spans with 25 channels [Fig. 10(a)],
(ii) two links with 15 channels that were added/dropped at the intermediate node [Fig. 10(b)].

For the first setup, 25 WDM channels with the pol-mux 16QAM format at 32 Gbaud, leading to a 200 Gbps data rate per channel were launched. We assumed a SNR threshold of 13.9 dB [1,6]. The wavelength spacing between the channels was assumed to be 50 GHz. We started with uniform 0 dBm of launch powers for all channels whose SNR values for each channel were measured/monitored by VPI. As stated above, VPI acted as the real-world/ground truth. Approximately 1 dB of maximum SNR difference between the untrained PLM and VPI was found, after setting equal the fiber parameters (dispersion coefficient, slope, attenuation coefficient of the fiber, nonlinearity coefficient, etc.). Then, with ML training (using the LM algorithm), the SNR mismatch was reduced to less than 0.1 dB (Fig. 2 of Section 3). This one-time trained PLM was then used with the power optimization algorithm. The algorithm converged to 199.31 dB and 7.34 dB for obj#1 and obj#2, depicted with the orange circles and dotted lines in Figs. 11(a) and 11(b), respectively.

We then optimized with the monitoring probes where the optimization algorithm was interfaced with the actual network (VPI). Again, the algorithm was run until it found the optimum for the objective function at hand. The algorithm converged to 204.96 dB and 7.64 dB for obj#1 and obj#2, depicted with the blue circles and dotted lines in Figs. 11(a) and 11(b), respectively. A relative mismatch of

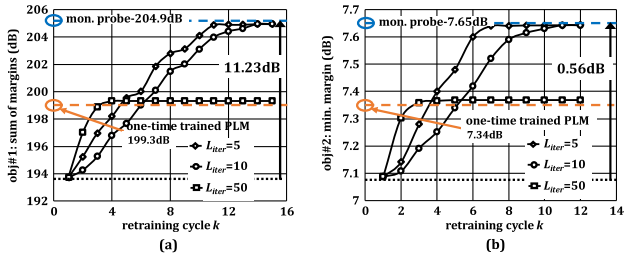

Fig. 11. (a) obj#1 and (b) obj#2 values as a function of the proposed DT approach retraining cycles (k), for flat EDFAs.

~5.6 dB for obj#1 and ~0.3 dB for obj#2 between the monitoring probes (optimum) and the one-time trained PLM was observed. With monitoring probes, the algorithm took around $L_{\text{mon\_prob}} \approx 120$ iterations to optimize the launch powers in both objectives. In case of the one-time trained PLM, the algorithm converged a bit faster, after $L_{\text{PLM}} \approx 90$ iterations.

We then optimized with our proposed PLM retraining/DT approach. We examined different PLM retraining periods of $L_{\text{iter}} = 5$, 10, and 50. Figure 11 shows the optimization objective values (evaluated in VPI) as a function of the retraining cycles (index k) of our proposed scheme. We see that with each training cycle, the objective value moves towards the optimum/reference obtained with the monitoring probes approach. For $L_{\text{iter}} = 5$, after approximately $k = 11$ iterations for obj#1 and $k = 8$ iterations for obj#2, the objective becomes nearly constant, indicating convergence. For small retraining periods, such as $L_{\text{iter}} = 5$, 10, the objective (both for obj#1 and obj#2) reached exactly that achieved with the monitoring probe-based approach, that is, $\text{obj}_{\text{DT}} = \text{obj}_{\text{mon\_prob}}$. However, for longer retraining periods, such as $L_{\text{iter}} = 50$, the algorithm converged near to the one-time trained PLM scheme. This happened because we allowed the optimization algorithm to perform long intermediate iterations without retraining the PLM. Within these iterations, the algorithm converged to an optimum; it could not improve its objective function ($\tilde{\text{obj}}_{\text{DT}}$) with the PLM used. However, after these long intermediate iterations, the PLM was not representing reality with good accuracy, and thus the corresponding real network objective (obj_{DT}) was rather suboptimal.

In reality, EDFA gains are not flat and come with ripples [6]. We assigned a gain ripple profile of p2p gain of 0.4 dB (±0.2 dB) to span the EDFAs. The PLM was then trained at 0 dBm of flat launch power and a maximum SNR difference of less than ±0.05dB was observed, as shown in Fig. 2(b), Section 3. This led to a mismatch of ~8.5 dB [Fig. 4(b)] and ~0.64 dB in the SNR margin between monitoring probes and one-time trained PLM optimization for obj#1 and obj#2, respectively. The mismatch is higher than previously, due to the EDFA gain ripples that make the physical layer more dynamic.

Figure 12(a) shows the power per channel (in dBm) at different retraining cycles k for a retraining period $L_{\text{iter}} = 5$ and for obj#1. Figure 12(b) shows the corresponding SNR (dB) values. For obj#1, with around $k = 20$ PLM retraining cycles, the transponders' launch power and SNR values converged near the optimal values obtained with the monitoring probe-based approach (Fig. 4). The algorithm reached obj#1 = 202.96 dB

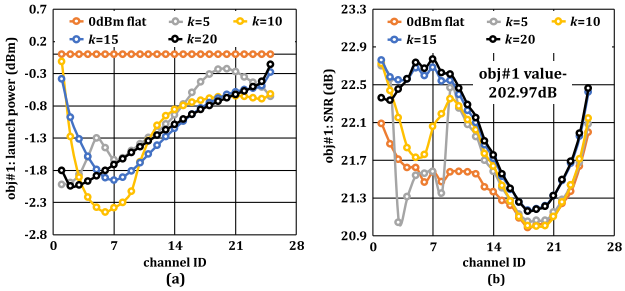

Fig. 12. (a) Launch power (dBm) and (b) SNR (dB) per channel as the number k of retraining cycles increases, for the proposed DT approach, $L_{\text{iter}} = 5$, obj#1, and EDFAs having p2p gain ripple of 0.4 dB.

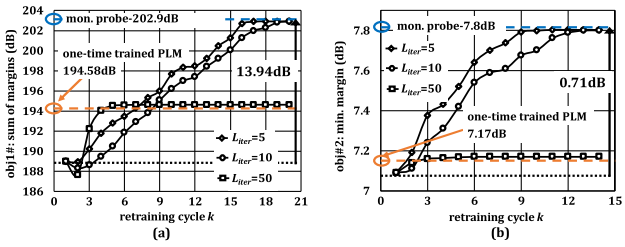

Fig. 13. (a) obj#1 and (b) obj#2 values as a function of the proposed DT approach retraining cycles (k), for EDFAs with p2p gain ripple of 0.4 dB.

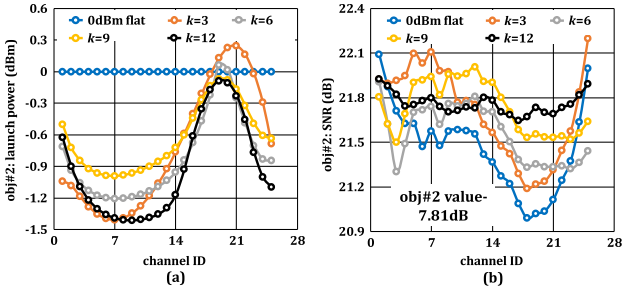

Fig. 14. (a) Launch power (dBm) and (b) SNR (dB) per channel as the number k of retraining cycles increases, for the proposed DT approach, $L_{\text{iter}} = 5$, obj#2, and EDFAs having p2p gain ripple of 0.4 dB.

[black curve in Fig. 13(a)], very close to the monitoring probe-based approach. Compared to the one-time trained PLM, it achieved an improvement of ∼8.4 dB, which is ∼0.34 dB of SNR improvement per channel.

Similar behavior was also observed for obj#2. Figures 14(a) and 14(b) show the input power (dBm) and the corresponding SNR (dB) for k retraining cycles with $L_{\text{iter}} = 5$ and for obj#2. From Fig. 13(b), it can be observed that after $K_{\text{retrain}} = 12$ retraining cycles, the algorithm reaches obj#2 = 7.8 dB, which is very close to that found with the monitoring probes approach. Compared to the one-time trained PLM, we achieved an improvement of ∼0.64 dB.

Figures 13(a) and 13(b) show the evolution of obj#1 and obj#2 with the proposed approach for different retraining periods of $L_{\text{iter}} = 5$, 10, 50. Note that we have comparatively higher savings (∼2.8 dB) with respect to the one-time

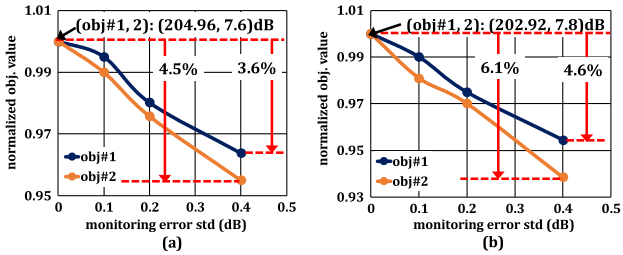

Fig. 15. Objective function variation as a function of the monitoring error std for (a) flat EDFA gain and (b) EDFA with gain ripple.

trained PLM compared to flat EDFAs (Fig. 11) because the physical layer is more dynamic with the rippled EDFAs. The small drop in the second retraining cycle of Fig. 13(a) can be explained by the PLM training process; in that cycle, the PLM did not match the real network very well and misled the optimization algorithm. This was then improved at the next retraining, which involved more monitoring information from the new/projected network state.

Monitors are not perfect and yield measurements that include errors, as discussed in Section 4. In our optimization, we assumed the use of SNR values measured from the coherent receivers, which are typically assumed to have good accuracy. However, there are some fast-varying impairments that result in SNR fluctuations and contribute to monitoring errors. A typical method to suppress those is to average the SNR measurements over a period longer enough than the frequency of such effects. Hence, we modeled these monitoring errors by adding a random GN v with mean = 0 and standard deviation (std) = 0.1, 0.2 and 0.4 dB, to the SNR values (provided by VPI). These noisy monitored SNR values were then fed to the interior point optimization algorithm and were reflected in the algorithm's derivative calculations.

The monitoring error results in a degraded optimization operation. In theory, the stochastic subgradient method [39] (which is a first-order method) with specific small steps can find the optimum for the assumed zero mean errors after a high number of iterations. However, such a method might not be applicable in real networks, where finite small steps are not feasible, and iterations must be constrained (to avoid effects of medium-term varying impairments). Also monitoring small SNR differences (due to the small steps) is rather hard (low slope of derivatives). So instead of seeking an *ideal* optimum, we focused on a more realistic case, and in the following results, we used the interior-point algorithm and a minimum power probe step $p_{\text{step}} = 0.1$ dBm.

Figure 15(a) shows the achieved objectives using the proposed DT approach with $L_{\text{iter}} = 5$ for varying monitoring error std (the mean was always equal to 0) for the flat EDFA case. Similar results were obtained for the monitoring probe optimization. A deterioration of the optimum with respect to ideal monitors (no noise, std = 0) was observed. For a std = 0.4 dB with flat EDFAs, the objective decreased by 3.6% and 4.5% for obj#1 and obj#2, respectively. For EDFAs with gain ripples, the related decrease was even higher, ∼5% and ∼6% for obj#1 and obj#2, respectively, as shown in Fig. 15(b).

We now turn our attention to the optimization time. As discussed, the PLM and the convex (interior-point) optimization

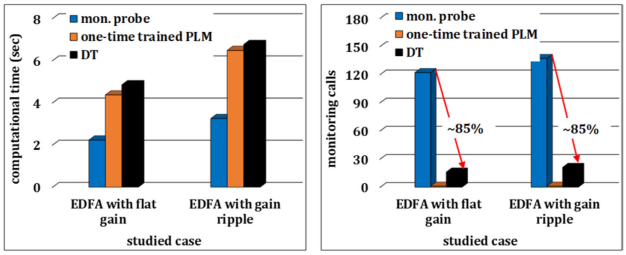

Fig. 16. (a) Computational time (s) and (b) number of monitoring calls required for EDFAs with flat and rippled gain cases, respectively.

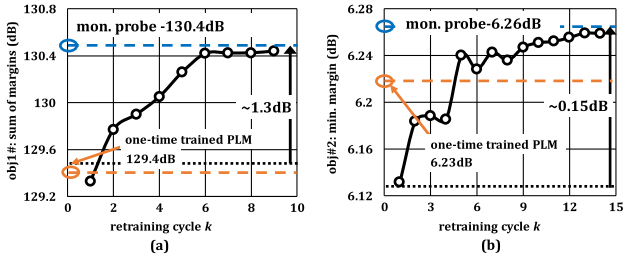

Fig. 17. (a) obj#1 and (b) obj#2 values as a function of the proposed DT approach retraining cycles (k), for flat EDFAs in the two-link setup.

algorithm were implemented in MATLAB. So, in MATLAB, we measured the overall computation time, which included the time t_{calc} that the algorithm calculated the gradients/Hessian and also the next launch powers, the time t_{train} to train with ML the PLM, and the time t_{PLM} for the SNR calculations by the PLM. Note that the PLM-related times appear only in the schemes that use it (one-time trained PLM and DT). Also, note that the computation time was measured until the algorithm obtained the optimum, so it included all retaining cycles, algorithm iterations, and finite-difference subroutine calls. Following the notation of Section 4, for the monitoring probes scheme we measured $L_{\text{mon\_prob}} \cdot t_{\text{calc}}$, for the one-time trained PLM we measured $t_{\text{train}} + L_{\text{PLM}} \cdot (D \cdot t_{\text{PLM}} + t_{\text{calc}})$, and for the DT we measured $K_{\text{retrain}} \cdot (t_{\text{train}} + L_{\text{iter}} \cdot (D \cdot t_{\text{PLM}} + t_{\text{calc}}))$. To obtain the DT results, we used a retraining period $L_{\text{iter}} = 5$.

Figure 16(a) shows the overall computational time for obj#1 for an EDFA with flat and rippled gain (std = 0). The computational time for the monitoring probe-based approach was the smallest, around 2 s for a flat gain EDFA and ~2.5 s for an EDFA with gain ripples, since it only includes t_{calc}. The one-time trained PLM was slightly faster than the DT approach (~4 s compared to ~4.5 for the flat EDFA gain case and ~7 s compared to ~7.5 for the case of an EDFA with gain ripple), since the PLM retraining time, which is their key difference, was quite fast. The computation time for the case of EDFAs with gain ripples [shown in Fig. 16(a)], was higher for all optimization schemes, since the algorithm took more iterations to reach the optimum.

From the above measurements, we excluded the monitoring time t_{mon}. It was excluded because we did not want to use some reference monitoring time. However, we expect it to be some orders of magnitude higher than the time scales reported in Fig. 16(a). Today, transponders report the SNR/BER every 15 min [38]. The reporting time can be substantially reduced down to seconds with NETCONF/YANG monitoring [12] or telemetry-based protocols [40,41]. However, such reporting periods target failure recovery use cases, which are substantially different from the power optimization use case studied in this paper. Moreover, we need to consider that in the monitoring probes scheme, the monitoring happens after the probing, that is, after changing the launch power of one or more connections. In such a case, we would have to wait for EDFA transient effects to settle [42]. Also, for optimization, we need to have a low monitoring error. So, we would need to time average to suppress the fast-varying impairment effects. Moreover, depending on the optimization method, it is required to monitor different times. In the one-time trained PLM approach, we monitor only once, at the beginning of the optimization. In the DT approach we monitor K_{retrain} times, once every PLM retraining cycle. In the monitoring probes approach, we train $L_{\text{mon\_prob}} \cdot D$. However, note that D depends on the optimization algorithm and the type of partial derivatives it calculates (first- or second-order). So we might have a different number of probes/monitoring per algorithm iteration.

To avoid any confusion with time scales, we show in Fig. 16(b) the number of monitoring calls, which in our simulations were measured as the number of times that we set new launch powers in VPI, executed the VPI transmission simulation, obtained the SNR values, and forwarded them in MATLAB. We can clearly see the substantially higher number of monitoring calls performed by the monitoring probes approach, which would result in substantially higher overall optimization time (the addition of computation [Fig. 16(a)] and monitoring [Fig. 16(b)] times).

Finally, to study a more network-like scenario, we extend the single-link setup to a two-link setup [Fig. 10(b)]. We established 15 connections with different add/drop locations and reused wavelengths at the intermediate node. We first focus on Case 1, the flat EDFA gain profiles. Figures 17(a) and 17(b) show the evolution of obj#1 and obj#2, respectively, as a function of the retraining cycles k for the DT scheme with $L_{\text{iter}} = 5$. The proposed DT approach improved by ~1.3 dB obj#1 and by ~0.15 dB obj#2, with respect to the one-time trained PLM scheme.

For Case 2, we assigned a gain ripple profile of ±0.2 dB and ±0.1 dB to link 1-2 and link 2-3, respectively. Figures 18(a) and 18(b) show the evolution of obj#1 and obj#2, respectively, as a function of the retraining cycles k for the DT with $L_{\text{iter}} = 5$ scheme. We obtained ~1.7 dB and ~0.6 dB of improvement for obj#1 and obj#2, respectively, with respect to training with a one-time trained PLM-based approach.

Compared to the single-link setup, the improvements obtained for the two-link setup were lower. The low channel load and the relatively wider channel separation due to ADD/DROP is one of the main reasons for the low value of improvements. Although we did not test bigger setups in VPI, since it is quite complicated, we believe that the benefits of our proposed solution are higher for a full network with higher load.

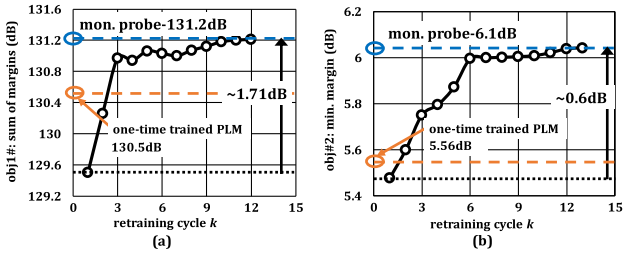

Fig. 18. (a) obj#1 and (b) obj#2 values as a function of the DT approach retraining cycles (k), for EDFAs with ripples in the two-link setup.

6. CONCLUSION

Despite the extensive literature on ML-based and ML-improved PLMs, there has been limited discussion on the use of PLMs in dynamic optimization. We proposed using an iterative closed control loop process to solve a complex/multivariable dynamic optimization problem. In particular, we proposed closing the control loop after several algorithm intermediate calculations, applying the intermediate calculated changes, and monitoring the outcome so that we realign (retrain) the PLM with the real-world/optical network. The PLM is used as a DT; it is applied to a network with evolving conditions, it is parametric and is dynamically adjusted so that it replicates the real world for the optimization at hand with enough accuracy. We applied our proposed method to solve the dynamic version of the launch power optimization problem. With the proposed PLM retraining/DT scheme, we showed an improvement of ~8.5 dB and ~0.64 dB in the sum of margins and the lowest margin, respectively, over optimization with a one-time trained PLM. Moreover, the proposed approach achieved near-to-optimum solutions, as found by optimizing and continuously probing and monitoring the network, but lowered the overall optimization time up to 85% by reducing the number of monitoring probes. We limited our study to 25 channels on a single link and 15 channels on a two-link network topology. However, we believe that our proposed solution is more beneficial once longer paths with heavy loads are considered.

Funding. Horizon 2020 Framework Programme (765275).

Acknowledgment. The authors would like to thank VPItransmissionMaker's support team for guidance in the receiver unit and end-to-end automation. This work is a part of the Future Optical Networks for Innovation, Research and Experimentation (ONFIRE) project (https://h2020-onfire.eu/), supported by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Actions.

REFERENCES

1. P. Soumplis, K. Christodoulopoulos, M. Quagliotti, A. Pagano, and E. Varvarigos, "Network planning with actual margins," J. Lightwave Technol. **35**, 5105–5120 (2017).
2. D. J. Ives, P. Bayvel, and S. J. Savory, "Adapting transmitter power and modulation format to improve optical network performance utilizing the Gaussian noise model of nonlinear impairments," J. Lightwave Technol. **32**, 4087–4096 (2014).
3. Y. Pointurier, "Design of low-margin optical networks," J. Opt. Commun. Netw. **9**, A9–A17 (2017).
4. J. L. Auge, "Can we use flexible transponders to reduce margins?" in *Optical Fiber Communication Conference (OFC)* (2013).
5. I. Sartzetakis, K. Christodoulopoulos, and E. Varvarigos, "Accurate quality of transmission estimation with machine learning," J. Opt. Commun. Netw. **11**, 140–150 (2019).
6. A. Mahajan, K. Christodoulopoulos, R. Martínez, S. Spadaro, and R. Muñoz, "Modeling EDFA gain ripple and filter penalties with machine learning for accurate QoT estimation," J. Lightwave Technol. **38**, 2616–2629 (2020).
7. I. Sartzetakis, K. Christodoulopoulos, and E. Varvarigos, "Cross-layer adaptive elastic optical networks," J. Opt. Commun. Netw. **10**, A154–A164 (2018).
8. E. Seve, J. Pesic, C. Delezoide, S. Bigo, and Y. Pointurier, "Learning process for reducing uncertainties on network parameters and design margins," J. Opt. Commun. Netw. **10**, A298–A306 (2018).
9. D. Rafique and L. Velasco, "Machine learning for network automation: overview, architecture, and applications [Invited Tutorial]," J. Opt. Commun. Netw. **10**, D126–D143 (2018).
10. S. Shahkarami, F. Musumeci, F. Cugini, and M. Tornatore, "Machine-learning-based soft-failure detection and identification in optical networks," in *Optical Fiber Communications Conference and Exposition (OFC)* (2018).
11. M. Hadi and E. Agrell, "Iterative configuration in elastic optical networks: (invited paper)," in *International Conference on Optical Network Design and Modeling (ONDM)* (2020).
12. K. Christodoulopoulos, C. Delezoide, N. Sambo, A. Kretsis, I. Sartzetakis, A. Sgambelluri, N. Argyris, G. Kanakis, P. Giardina, G. Bernini, D. Roccato, A. Percelsi, R. Morro, H. Avramopoulos, P. Castoldi, P. Layec, and S. Bigo, "Toward efficient, reliable, and autonomous optical networks: the ORCHESTRA solution [Invited]," J. Opt. Commun. Netw. **11**, C10–C24 (2019).
13. M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," Synth. Lect. Commun. Netw. **3**, 1–211 (2010).
14. M. Channegowda, R. Nejabati, and D. Simeonidou, "Software-defined optical networks technology and infrastructure: enabling software-defined optical network operations [Invited]," J. Opt. Commun. Netw. **5**, A274–A282 (2013).
15. S. Yan, F. N. Khan, A. Mavromatis, D. Gkounis, Q. Fan, F. Ntavou, K. Nikolovgenis, F. Meng, E. H. Salas, C. Guo, C. Lu, A. P. T. Lau, R. Nejabati, and D. Simeonidou, "Field trial of machine-learning-assisted and SDN-based optical network planning with network-scale monitoring database," in *European Conference on Optical Communication (ECOC)* (2017).
16. L. Velasco, A. C. Piat, O. Gonzlez, A. Lord, A. Napoli, P. Layec, D. Rafique, A. D'Errico, D. King, M. Ruiz, F. Cugini, and R. Casellas, "Monitoring and data analytics for optical networking: benefits, architectures, and use cases," IEEE Netw. **33**, 100–108 (2019).
17. N. Stojanovic and D. Milenovic, "Data-driven digital twin approach for process optimization: an industry use case," in *IEEE International Conference on Big Data (Big Data)* (2018).
18. L. Wright and S. Davidson, "How to tell the difference between a model and a digital twin," Adv. Model. Simul. Eng. Sci. **7**, 13 (2020).
19. M. Grieves, "Digital twin: manufacturing excellence through virtual factory replication," White paper, 2014, https://www.3ds.com/fileadmin/PRODUCTS-SERVICES/DELMIA/PDF/Whitepaper/DELMIA-APRISO-Digital-Twin-Whitepaper.pdf.
20. E. Glaessgen and D. Stargel, "The digital twin paradigm for future NASA and US Air Force vehicles," in *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA* (2012).
21. K. D. R. Assis, S. Peng, R. C. Almeida, H. Waldman, A. Hammad, A. F. Santos, and D. Simeonidou, "Network virtualization over elastic optical networks with different protection schemes," J. Opt. Commun. Netw. **8**, 272–281 (2016).
22. J. Cho, S. Chandrasekhar, E. Sula, S. Olsson, E. Burrows, G. Raybon, R. Ryf, N. Fontaine, J. Antona, S. Grubb, P. Winzer, and A. Chraplyvy, "Maximizing fiber cable capacity under a supply power constraint using deep neural networks," in *Optical Fiber Communications Conference and Exhibition (OFC)* (2020).

23. P. Soumplis, K. Christodoulopoulos, and E. Varvarigos, "Dynamic connection establishment and network re-optimization in flexible optical networks," Photon. Netw. Commun. **29**, 307–321 (2015).
24. A. Castro, L. Velasco, M. Ruiz, M. Klinkowski, J. P. Fernández-Palacios, and D. Careglio, "Dynamic routing and spectrum (re)allocation in future flexgrid optical networks," Comput. Netw. **56**, 2869–2883 (2012).
25. P. Papanikolaou, K. Christodoulopoulos, and E. Varvarigos, "Incremental planning of multi-layer elastic optical networks," in *International Conference on Optical Network Design and Modeling (ONDM)* (2017).
26. D. J. Ives and S. J. Savory, "Transmitter optimized optical networks," in *Optical Fiber Communication Conference (OFC)* (2013).
27. P. Poggiolini, G. Bosco, A. Carena, V. Curri, Y. Jiang, and F. Forghieri, "The GN-model of fiber non-linear propagation and its applications," J. Lightwave Technol. **32**, 694–721 (2014).
28. H. Rabbani, L. Beygi, S. Ghoshooni, H. Rabbani, and E. Agrell, "Quality of transmission aware optical networking using enhanced Gaussian noise model," J. Lightwave Technol. **37**, 831–838 (2019).
29. I. Roberts, J. M. Kahn, and D. Boertjes, "Convex channel power optimization in nonlinear WDM systems using Gaussian noise model," J. Lightwave Technol. **34**, 3212–3222 (2016).
30. I. Roberts and J. M. Kahn, "Measurement-based optimization of channel powers with non-Gaussian nonlinear interference noise," J. Lightwave Technol. **36**, 2746–2756 (2018).
31. D. Bertsekas, *Dynamic Programming and Stochastic Control*, 1st ed. (Elsevier/Academic, 1976).
32. R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. (MIT Press, 1998).
33. O. Devolder, F. Glineur, and Y. Nesterov, "First-order methods of smooth convex optimization with inexact oracle," Math. Program. **146**, 37–75 (2014).
34. https://www.vpiphotonics.com/index.php.
35. H. P. Gavin, "The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems," Notes for CE281 (Duke University, 2020), http://people.duke.edu/~hpgavin/ce281/lm.pdf.
36. S. Boyd and J. Park, "Subgradient methods," Notes for EE364b (Stanford University, 2014), https://stanford.edu/class/ee364b/lectures/subgrad_method_notes.pdf.
37. M. S. Gockenbach, "Computing derivatives by finite differences," Notes for MA5630 (Michigan Technological University, 2013), https://pages.mtu.edu/~msgocken/ma5630spring2003/lectures/diff/diff/diff.html.
38. https://metro-haul.eu/deliverables/.
39. S. Boyd, A. Mutapcic, and J. Duchi, "Stochastic subgradient methods," Notes for EE364b (Stanford University, 2014), https://web.stanford.edu/class/ee364b/lectures/stoch_subgrad_notes.pdf.
40. R. Vilalta, N. Yoshikane, R. Casellas, R. Martínez, S. Beppu, D. Soma, S. Sumita, T. Tsuritani, I. Morita, and R. Muñoz, "GRPC-based SDN control and telemetry for soft-failure detection of spectral/spacial superchannels," in *European Conference on Optical Communication (ECOC)* (2019).
41. F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi, "Network telemetry streaming services in SDN-based disaggregated optical networks," J. Lightwave Technol. **36**, 3142–3149 (2018).
42. C. Tian and S. Kinoshita, "Analysis and control of transient dynamics of EDFA pumped by 1480- and 980-nm lasers," J. Lightwave Technol. **21**, 1728–1734 (2003).