DeepLS: Local Search for Network Optimization based on Lightweight Deep Reinforcement Learning

Nicola Di Cicco, *Graduate Student Member, IEEE*, Memedhe Ibrahimi, *Member, IEEE*
Sebastian Troia, *Member, IEEE*, and Massimo Tornatore, *Fellow, IEEE*

***Abstract*—Deep Reinforcement Learning (DRL) is being investigated as a competitive alternative to traditional techniques for solving network optimization problems. A promising research direction lies in enhancing traditional optimization algorithms by offloading low-level decisions to a DRL agent. In this study, we consider how to effectively employ DRL to improve the performance of Local Search algorithms, i.e., algorithms that, starting from a candidate solution, explore the solution space by iteratively applying local changes (i.e., moves), yielding the best solution found in the process. We propose a Local Search algorithm based on lightweight Deep Reinforcement Learning (DeepLS) that, given a neighborhood, queries a DRL agent for choosing a move, with the goal of achieving the best objective value in the long term. Our DRL agent, based on permutation-equivariant neural networks, is composed by less than a hundred parameters, requiring only up to ten minutes of training and can evaluate problem instances of arbitrary size, generalizing to networks and traffic distributions unseen during training. We evaluate DeepLS on two illustrative NP-Hard network routing problems, namely OSPF Weight Setting and Routing and Wavelength Assignment, training on a single small network only and evaluating on instances 2x-10x larger than training. Experimental results show that DeepLS outperforms existing DRL-based approaches from literature and attains competitive results with state-of-the-art metaheuristics, with computing times up to 8x smaller than the strongest algorithmic baselines.**

***Index Terms*—Deep Reinforcement Learning, Local Search, IP Networks, Optical Networks, NP-Hard Optimization**

I. Introduction

Machine Learning (ML) has become a widely adopted decision-support tool in communications networks. Remarkable results have been achieved in many applications, such as in traffic prediction [1], failure detection [2], traffic engineering [3], link load control [4], and routing and spectrum assignment problems in optical networks [5], [6].

As the size and the complexity of network optimization problems are bound to grow over time, new solving methods that are both computationally scalable and that can produce good-quality solutions are currently sought for several networking domains, such as in IP and in optical networks. Among the different ML paradigms, Reinforcement Learning (RL) is currently attracting significant attention. What makes RL particularly attractive is its capability, unlike Supervised Learning (SL), to learn without labelled data. The high-level objective of RL applied to NP-Hard optimization problems is to learn specialized heuristic algorithms tailored for specific problem classes (e.g., learning a constructive heuristic for static Routing and Wavelength Assignment (RWA) in optical networks [6]). Following this line of reasoning, RL-based heuristics have been applied with success to several classical problems in the Operations Research (OR) literature, such as the Knapsack Problem, the Bin-Packing Problem, the Travelling Salesman Problem, and the Capacitated Facility Location Problem [7]–[10].

Generally speaking, Bengio et al. [11] have laid out three high-level paradigms for ML applied to combinatorial optimization: *i)* end-to-end learning, *ii)* parameter configuration, and *iii)* ML alongside OR. In end-to-end learning, a ML black-box model outputs a feasible solution for a given problem instance. In parameter configuration, a ML model predicts the optimal setup parameters of a given OR algorithm, for instance a solver or a metaheuristic, for a given problem instance. Finally, in ML alongside OR, one or more ML-based algorithms replace lower-level functions that were originally implemented by an OR algorithm (e.g., choosing the best variable to branch in each Branch-and-Bound iteration [10]).

Among the aforementioned paradigms, ML alongside OR (ML+OR) stands out, as it preserves the logical structure of well-established optimization strategies, while leveraging ML to address their shortcomings. For example, many low-level components in well-known heuristics are often arbitrarily defined, such as the criteria for choosing a move in Local Search or the mutation/crossover functions in a Genetic Algorithm. As an illustrative recent research achievement, the winner of a recent competition on solving the Vehicle Routing Problem with Time Windows [12] employed ML+OR methodologies. Therefore, we argue that investigating ML+OR methodologies for solving classical optimization problems in networking is a research topic of great interest. Our research questions are therefore as follows: *can we enhance already existing network optimization algorithms by offloading selected tasks to (possibly lightweight) learning intelligence? What are the gains compared with handcrafted state-of-the-art algorithms?*

To demonstrate the potential of ML+OR for network optimization, we propose DeepLS, a simple Local Search (LS) algorithm augmented with Deep Reinforcement Learning. We apply DeepLS for solving two illustrative NP-Hard routing problems in communication networks, namely, the OSPF Weight Setting problem (OWS) and the Routing and Wavelength Assignment problem (RWA). We chose OWS and RWA since they are both well-studied problems in communications literature for which strong algorithmic baselines are available, and are often solved as subproblems in high-level network

Nicola Di Cicco, Memedhe Ibrahimi, Sebastian Troia, and Massimo Tornatore are with the Department of Electronics, Information and Bioengineering (DEIB), Politecnico Di Milano, Italy. E-mail: {name}.{surname}@polimi.it

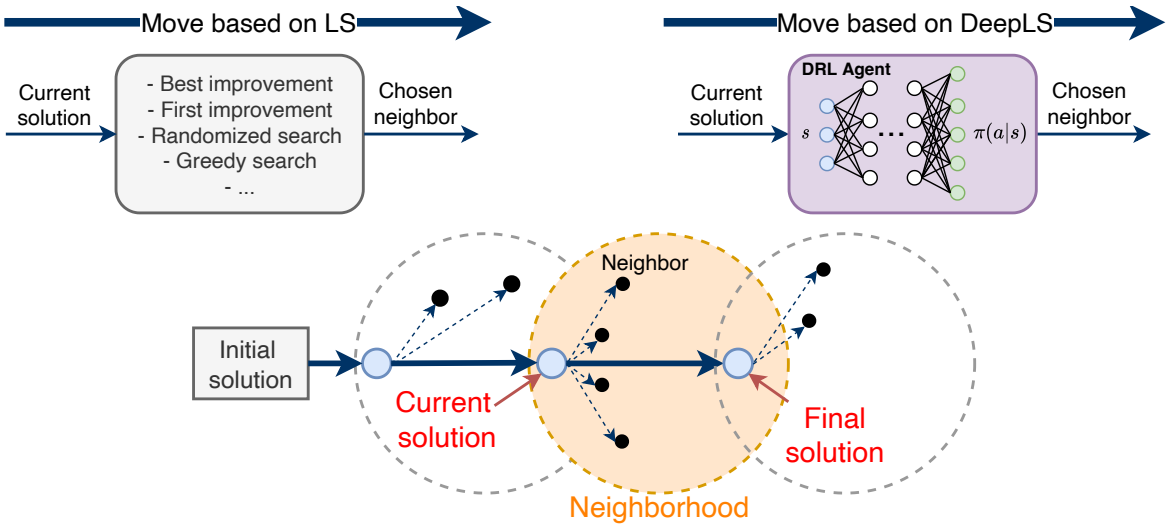


Fig. 1. Local Search (LS) vs. Deep Reinforcement Learning-augmented Local Search (DeepLS). Given an initial solution, traditional Local Search algorithms iteratively explore the search space following deterministic policies. However, designing the best neighbourhood choice policy for a given problem family is not straightforward. As such, the choice of a neighbour can be offloaded to a lightweight DRL agent, with the goal of learning a smart neighbourhood selection policy tailored for the specific optimization problem that is being solved.

management frameworks.

DeepLS leverages a DRL agent for guiding the LS neighbor selection procedure, with the goal of achieving better solutions in the long-term. Specifically, in Fig. I we illustrate how DRL can be used to augment a standard LS procedure: instead of applying myopic decision criteria, a DRL agent is trained to select the sequence of neighbours leading to the best final solution. To this end, we employ a lightweight permutation-equivariant artificial neural network architecture, composed of less than a hundred parameters, that can perform inference for problem instances of arbitrary size. As DRL agents often require a large number of environment interactions for learning an effective policy, we devise two fundamental countermeasures for shortening the required training times: i) we train our DRL agent only on one small network topology, which is significantly less complex to simulate, and ii) we train our DRL agent only for a small number of LS iterations, where it can at best converge to a local minima close to the starting solution. This allows to dramatically reduce training times down to few minutes, compared to the tens of hours required for state-of-the-art DRL-based approaches proposed in previous literature. In terms of solution quality, our approach outperforms both state-of-the-art DRL-based approaches and is competitive with handcrafted metaheuristics. To summarize, our results show that a simple Local Search heuristic, if augmented with lightweight DRL, can compete with far more elaborated handcrafted algorithms requiring minimal training effort and inference times overhead. These insights open up exciting research directions in many applications of Machine Learning for network optimization.

The remainder of this paper is organized as follows. In Section II we outline related works on RL applied to optimization problems in communications networks. In Section III we provide essential background notions on Reinforcement Learning and Deep Reinforcement Learning. In Section IV we outline our proposed Local Search methodology, framing it into a Markov Decision Process. We illustrate the design of our lightweight neural architecture, and we outline its application to the OSPF Weight Setting (OWS) problem and the Routing and Wavelength Assignment (RWA) problem in optical networks. In Section V we illustrate our experimental results on the OWS and RWA problems. In Section VI we outline our conclusions and future research directions.

II. Related Work

In this Section we briefly survey recent works in the field of RL applied to optimization in communications network. Moreover, we outline recent progress on the application of RL for enhancing Local Search algorithms.

A. Reinforcement Learning for Network Optimization

RL applied to optimization and control of communication networks has received significant attention in the recent years. Since RL is a universal methodology for data-driven sequential decision making, many applications have been explored in the networking literature. In the following, we overview recent studies on RL applied to communication networks.

In [3], a DRL agent based on a Graph Neural Network (GNN) is used for learning a heuristic for the Multi-Commodity Flow (MCF) problem. The DRL agent learns the OSPF weights optimizing the min-max link load in the network. The proposed approach attains competitive results to a state-of-the-art optimizer [13] in significantly shorter computing times. The DRL agent can be applied to graphs of any size, generalizing to topologies not seen during training.

In [14], a DRL agent is used to learn a heuristic for the MCF problem. The DRL agents learn edge weights that are then converted to per-flow splitting ratios via "SoftMin routing". The work in [15] extends [14] with the use of GNNs. In particular, authors emphasize on the capability of GNNs

to perform inference and generalize on graphs of arbitrary size. Both frameworks require solving the associated Linear Program of the MCF problem at every learning iteration, hence optimal solutions may not be attainable for larger problem instances or for NP-Hard optimization problems.

In [5], DRL is used to learn a heuristic algorithm for dynamic Routing, Modulation and Spectrum Assignment (RMSA) in optical networks. Given a connection request, the DRL agent either selects one among the K pre-computed paths and a feasible spectrum allocation or issues a proactive rejection. The proposed approach outperforms baseline greedy heuristics such as Shortest-Path First-Fit and K-Shortest-Paths First-Fit, but no comparison is provided against more competitive metaheuristics.

In [6], DRL is used to learn a heuristic algorithm for static Routing and Wavelength Assignment (RWA). The DRL agents operates similarly as in [5], but countermeasures are taken for large problem instances for which the sparsity of the reward hindered the final performance. The proposed approach outperforms baseline greedy heuristics and is competitive with a state-of-the-art Genetic Algorithm tailored for RWA, with significant savings in computing times.

In [16], DRL is used to learn a slice admission control policy in a 5G Radio Access Network (RAN). The objective is to maximize the revenue of the operator considering low and high priority slices, balancing penalties coming from rejecting a slice or failing to scale an already admitted slice due to resource unavailability. The proposed approach outperforms both static and threshold-based heuristics.

B. Reinforcement Learning alongside Local Search

Recently, several works have investigated the use of RL alongside Local Search for solving NP-Hard optimization problems. Indeed, applying RL to further improve general-purpose and well-performing heuristics is gathering attention in the research community.

In [17], DRL is used alongside Simulated Annealing for solving a maintenance planning problem. DRL is used to learn a Local Search algorithm that iteratively refines the best solution found by each iteration of SA. The proposed approach outperforms baseline SA and other data-driven heuristics.

In [7], DRL is used alongside Simulated Annealing for solving classical combinatorial optimization problem, such as the Knapsack Problem, the Bin Packing Problem and the Travelling Salesman Problem. DRL is used to learn the proposal function that is internally called by SA at each iteration. The proposed approach, albeit being a general-purpose solution method, attains competitive results to specialized heuristics for the considered problems.

In [18], a DRL-based Iterated Local Search was developed for solving the additive manufacturing machine scheduling problem. In particular, the Local Search procedure was implemented as a Variable Neighbourhood Search algorithm, for which the DRL agent selects the best neighbourhood to explore at each time-step. The proposed approach outperforms an evolutionary algorithm on medium to large problem instances.

C. Contributions

In the context of ML for Network Optimization, previous literature mostly focused on end-to-end learning [2], [5], [6], [14]. Compared to these works, we focus on how to augment an existing optimization method, e.g., a Local Search algorithm, with lightweight DRL-based intelligence.

In the context of DRL applied to Local Search algorithms, differently from [17] and [18], the DRL agent in our proposed solution learns *to choose the best move in a neighbourhood* rather than *to choose a specific neighbourhood structure*. As in [7], we use a Deep Set-like architecture [19] for implementing our neural network, but we did not opt for an augmented SA. From preliminary results, we have found the probabilistic rejections of SA to be detrimental for learning. We speculate that with simple LS, the DRL agent can learn by itself to balance between exploration and exploitation, and that the probabilistic rejections of SA may inadvertently pollute exploration. Moreover, since the Markov chains traversed by SA are notoriously very long and networking environments may be cumbersome to simulate, we concluded that Simulated Annealing as a methodology is not compatible with our requirements of short computational times.

In our work, we improve over the state-of-the-art by dealing with several research challenges of practical importance for a successful integration of ML in network optimization:

- **Training times**: DRL is notorious for being extremely sample inefficient, often requiring hours-long training before converging to a reasonable policy. DeepLS exploits a smart training procedure that allows to reach convergence in few minutes of training.
- **Generalization capabilities**: generally speaking, algorithms should work well for a broad and diversified set of problem instances. DeepLS shows remarkable generalization capabilities: while being trained on only one small network topology, it shows remarkable generalization capabilities to larger network topologies and traffic matrices unseen at training time.
- **Computational times**: Integrating Deep Neural Networks inside an optimization algorithm might introduce a significant overhead in the overall computational time. DeepLS leverages a lightweight neural network of less than a hundred parameters, therefore introducing negligible additional complexity in the Local Search procedure. Depending on the complexity of the optimization problem, computational times range from few seconds to few tens of seconds, which are fully in-line with what it is expected from state-of-the-art metaheuristics.
- **Solution quality**: DeepLS outputs solutions that outperform previous state-of-the-art DRL-based algorithms and either match or outperform significantly more complex handcrafted metaheuristics. In particular, the most relevant related work to ours is [3], in which authors propose an iterative DRL-based procedure for setting OSPF weights that leverages Graph Neural Networks (GNNs). Compared to [3], we achieve better performance with significantly shorter training/inference times, and without the need for a complex GNN model.

III. Background

In this section we provide essential theoretical background on RL and DRL. Moreover, we outline the main definitions and properties of permutation-equivariant neural networks.

A. Reinforcement Learning

RL is a subset of ML which targets sequential decision-making problems. An agent interacts with a dynamic environment by taking actions, with the goal of maximizing the accumulation of rewards over a time horizon [20]. Formally, an RL environment is often modelled as a Markov Decision Process (MDP). An MDP is represented by the tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability matrix, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and $\mu : \mathcal{S} \to [0, 1]$ is the set of initial probabilities. We define the discounted return at time step t as $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k$.

The goal of an RL agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted accumulation of rewards over a (possibly infinite) time-horizon T, as follows:

$$\arg\max_{\pi} J(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t \;\middle|\; \pi \right] \tag{1}$$

Given a policy π, we define its value function $V_\pi : \mathcal{S} \to \mathbb{R}$ as $V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$, and its action-state value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as $Q(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$.

In the following, we will consider episodic RL settings in which the time horizon T is assumed to be finite. For example, in RL applied to optimization, we can consider an episode to be corresponding to a single problem instance. In such episodic settings, the discount factor γ in Eq. (1) is often dropped, leading to the optimization of the undiscounted sum of rewards over an episode [21]. The shape of the reward function is a critical design choice in RL, as it ultimately defines the problem we are solving. Interestingly, whether or not "reward is enough" to achieve every conceivable optimization goal is currently an open research question [22].

In many applications of practical interest, the number of states and actions in an MDP may either be combinatorial or infinite. For instance, decision variables in an Integer Linear Programming (ILP) optimization problem may take a combinatorial number of possible values. Because of this it may not be possible, due to time and memory constraints, to represent and solve the MDP via exact methods. As such, we need to leverage a function approximation for solving complex MDPs via RL. In particular, the seminal paper of Mnih et al. [23] illustrated that it is possible to learn to play Atari games via Reinforcement Learning by using deep convolutional neural networks as function approximators, giving rise to the field of DRL. Since then, significant results have been achieved in a plethora of complex tasks using DRL, from continuous control to playing modern real-time strategy games [24]–[26].

B. Permutation-equivariant neural networks

A generic algorithm for network optimization operates on sets of decision variables. As such, to apply DRL to network optimization, we need to leverage neural network architectures that can satisfy two fundamental properties. First, we want a neural network able to perform inference from an arbitrary number of inputs. This is because we want to apply a trained model on problem instances different (and possibly larger) than the ones seen during training. Second, we want the outputs of the neural network to be equivariant with respect to the ordering of its inputs. Indeed, even if we permute the ordering (i.e., the indexing) of the decision variables in an optimization problem, the problem that is being represented remains the same. Formally, a multi-output neural network is permutation-equivariant if the following holds:

$$\mathbf{f}_\theta(\text{perm}(\mathbf{x})) = \text{perm}(\mathbf{f}_\theta(\mathbf{x})) \tag{2}$$

where $\mathbf{f}_\Theta(\cdot)$ is the neural network parameterized by θ, $\mathbf{x}$ is the vector of inputs and $\text{perm}(\cdot)$ is a permutation operator. In practice, if we apply a permutation on the inputs, the same permutation is reflected in the outputs. For example, in the context of optimization problems, each input may correspond to a decision variable, and outputs may correspond to scores for each variable. As such, for the scores to be consistent, the neural network needs to be permutation equivariant.

In [19] authors outline general principles for designing permutation equivariant neural network layers. A permutation-equivariant layer can be generally expressed as follows:

$$\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{\Lambda} + \mathbf{1}\mathbf{1}^\top \mathbf{x}\mathbf{\Gamma}) \tag{3}$$

where $\mathbf{\Lambda}, \mathbf{\Gamma} \in \mathbb{R}^{m \times l}$ are learnable parameters, $\mathbf{x} \in \mathbb{R}^{n \times m}$ are the inputs, and $\sigma(\cdot)$ is an element-wise non-linearity. In particular, the input matrix $\mathbf{x}$ represents n elements characterized by m features each. The transformations $\mathbf{x}\mathbf{\Lambda}$ and $\mathbf{x}\mathbf{\Gamma}$ are permutation equivariant, since they are equivalent to applying the linear transformation $\mathbf{\Lambda}$ and $\mathbf{\Gamma}$, respectively, to each row of $\mathbf{x}$. Therefore, if a permutation is applied to the input elements $\mathbf{x}$, the same permutation is reflected on the elements of $\mathbf{f}(\mathbf{x})$. Stacking multiple layers defined as Eq. (3) allows to build permutation-equivariant deep neural networks.

Algorithm 1 DRL-Augmented Local Search (DeepLS)

Require: Objective function $O(\cdot)$, starting solution s_0 (e.g., via a greedy heuristic), neighbour selection policy $\pi(a|s)$, transition function $\text{Step}(\cdot)$

$s_{\text{best}} \leftarrow s_0$
$O_{\text{best}} \leftarrow O(s_0)$
for $t \leftarrow 0$ to $N_{\text{iter}} - 1$ **do**
  $a_t \leftarrow \pi(a|s = s_t)$
  $s_{t+1} \leftarrow \text{Step}(s_t, a_t)$
  if $O(s_{t+1}) < O_{\text{best}}$ **then**
    $s_{\text{best}} \leftarrow s_{t+1}$
  end if
end for
return s_{best}

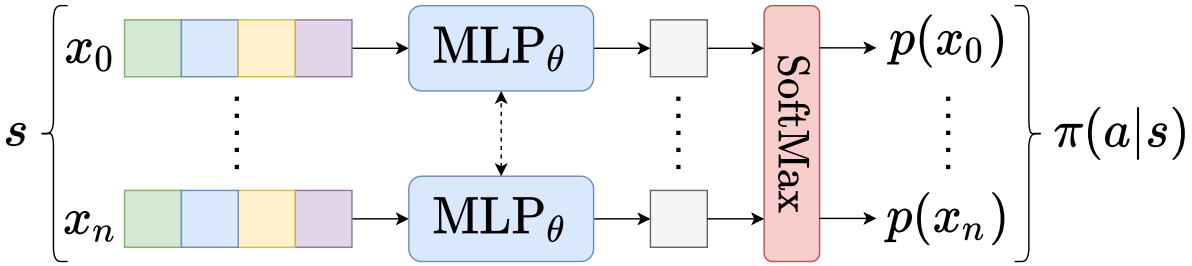


Fig. 2. DeepLS policy network architecture for a discrete action space. An action consists in choosing one among the n decision variables. Each decision variable $x_0, \ldots, x_n$ is associated with a set of features (e.g., if we associate one decision variable to a link in a network topology, link features can be the current link load, the current link weight, and so on). Features are processed in parallel by the same single-output neural network. Individual outputs are then aggregated (e.g., via SoftMax) to produce the policy $\pi(a|s)$.

IV. DeepLS: DRL-Augmented Local Search

Our proposed DRL-Augmented Local Search algorithm (DeepLS) is outlined in Algorithm 1. At each execution step, the neighbour selection policy implemented by a DRL agent $\pi(a|s)$, chooses a move by observing only the current solution. The Step function updates the current solution according to the move chosen by the policy. After N_{iter} iterations, the best solution found is returned. The transitions from a solution to the next one depend only on the current solution, therefore this Local Search procedure realizes a Markov chain over the solution space. We extend this Markov chain to an episodic MDP, which we aim to solve via model-free RL. In particular, in the following we will refer to a full execution of Algorithm 1 for some problem instance as an episode.

In the remainder of this Section, we outline our lightweight neural network architecture for implementing the DRL agent $\pi(a|s)$, and we illustrate its application to the OWS and RWA problems.

A. Neural network architecture

An ML-based solution method for solving optimization problem should have the capability *to generalize* to problem instances unseen during training [11]. In the context of network optimization, we are particularly interested in neural architectures enabling inference to network topologies of arbitrary size, possibly larger than the ones seen during training. Previous works in [3], [15] propose algorithms based on Graph Neural Networks (GNNs) for solving traffic engineering problems. Though we share the intuition behind the use of GNNs in the context of communication networks, in this work we propose a simpler alternative based solely on Multi-Layer Perceptrons (MLPs). In particular, as illustrated in [19] and suggested in [7], we employ an MLP-based architecture that can perform inference to instances of arbitrary size and is equivariant with respect to the ordering of the inputs. Our architecture sidesteps the complexity of the iterative message-passing procedure of GNNs[1].

[1]In GNNs, to ensure that each node receives information from every other node, one needs a number of message-passing iterations equal to the network diameter. In sparse graphs (e.g., an optical network), GNNs with several hidden layers may suffer from issues such as oversmoothing [27] and oversquashing [28]. Moreover, large inference times in deep GNNs are among the most critical research challenges in the graph learning community [29].

We consider a neural architecture operating on sets of decision variables (e.g., the weights associated to each link in a network topology). Let n be the number of decision variables in our problem instance. We associate m features to each decision variable, hence, a state s is a matrix $s \in R^{n \times m}$. Let MLP_θ be an MLP neural network parameterized by θ with m inputs and one output. We can therefore process in parallel, via batching, all decision variables' features via the same single-output neural network MLP_θ. With reference to Eq. (3), our architecture is equivalent to stacking multiple layers of the form $\mathbf{o}^{l+1} = \sigma(\mathbf{o}^l \mathbf{\Lambda_l})$, where o^l and Λ_l are the parameters and the outputs of the l-th MLP layer, respectively. We did not find the additional transformation Θ to benefit performance, and we opted for keeping the number of parameters to the minimum. The policy $\pi(a|s)$ can be therefore derived by aggregating the outputs of each forward pass in MLP_θ, e.g., via SoftMax.

A schematic representation of a policy network for a discrete action space, i.e., where the policy $\pi(a|s)$ defines a categorical distribution, is illustrated in Fig. 2. In particular, an action corresponds to choosing one among the decision variables. Outputs from each forward pass of MLP_θ are considered as non-normalized logits, which are then transformed via the SoftMax operator. The outputs of the SoftMax define a categorical distribution over the decision variables, realizing the discrete action space policy $\pi(a|s)$. Since each decision variable shares the same MLP_θ, the realized policy $\pi(a|s)$, i.e., the SoftMax aggregation over each individual output from MLP_θ, is permutation-equivariant with respect to the input decision variables. This fundamental property makes it possible to support arbitrily large state and action spaces, which allows to apply the same pre-trained policy to problem instances of arbitrary size without the need for retraining.

Actor-Critic DRL algorithms (e.g., PPO [25]) require both an "Actor" network, parameterizing the policy $\pi(a|s)$, and a "Critic" network, parameterizing the value function $V(s)$. We can use the same architecture of Fig. 2 for implementing the Critic network. For a discrete action space, we consider each single output of MLP_θ to represent an action-value function $Q_\phi(s, a)$. From the action-value function, from which we can compute the value function $V_\phi(s) = \mathbb{E}_{\pi_\theta(a|s)}[Q_\phi(s, a)]$.

Thanks to fast batch parallelization, this neural architecture can be very efficiently applied to problem instances of arbitrary

size. Moreover, as outlined in Section III-B, this architecture is permutation equivariant with respect to the ordering of its inputs, i.e., the decision variables. We highlight that our architecture resembles the "Readout" module placed before the final outputs of a GNN [30].

At evaluation time we aim to assess the generalization capabilities of DeepLS to problem instances unseen during training. In particular, with our experimental results, we quantitatively answer the following research questions:

1) Can our approach generalize to traffic distributions unseen during training?
2) Can our approach generalize to larger network topologies unseen during training?
3) Can our approach, if trained on short episodes (i.e., executions of Algorithm 1 where N_{iter} is small) generalize to longer episodes, therefore achieving better solutions in the long run?

B. DeepLS for OSPF Weight Setting

1) Premise on the Multi-Commodity Flow problem (MCF): optimal traffic engineering can be achieved by solving the well-known Multi-Commodity Flow (MCF) problem. The MCF problem can be formulated as a Linear Programming (LP) optimization problem, hence it can be solved in polynomial time. Let $G(V, E)$ be a bidirectional graph where each edge $(u, v) \in E$ has capacity $c_{u,v}$. Let $d_{s,t}$ be the flow units that need to be routed from the source node $s \in V$ to the destination node $t \in V$. We define continuous decision variables $f^t_{u,v}$, representing the amount of flow with destination $t \in V$ routed on edge $(u, v) \in E$. As objective function, we consider the minimization of the maximum normalized link load. The LP formulation of the MCF problem reads as follows:

$$\min \max_{(u,v)\in E} \sum_{t\in V} \frac{f^t_{u,v}}{c_{u,v}} \tag{4}$$

$$\sum_{t\in V} f^t_{u,v} \leq c_{u,v} \quad \forall (u, v) \in E \tag{5}$$

$$\sum_{v\in V} f^t_{s,v} - \sum_{u\in V} f^t_{u,s} = d_{s,t} \quad \forall s, t \in V,\ s \neq t \tag{6}$$

$$f^t_{u,v} \geq 0 \quad \forall t \in V,\ \forall (u, v) \in E \tag{7}$$

Objective function (4) minimizes the maximum normalized link load, Eq. (5) are the maximum link capacity constraints, and Eq. (6) are the flow conservation constraints. Being this problem an LP, its optimal solution can be computed very efficiently by commercial solvers.

2) The OSPF Weight Setting problem (OWS): let us now consider the case in which routing configurations are determined by link-state protocols, such as OSPF, in which routes are computed using Dijkstra's algorithm on the weighted network topology graph. In this context, the problem of determining the edge weights realizing the optimal routing of (4)-(7) is NP-Hard [31]. We therefore consider the problem of finding the OSPF weights that minimize the maximum link load in the network (4). We also consider Equal Cost Multi-Path (ECMP) routing, i.e., flows directed to the same destination are equally distributed among all shortest paths of equal cost.

3) Formulating LS for OWS as an MDP: we consider an LS algorithm where, at each iteration, the weight of one link is increased and a new OSPF routing is computed.

State Space: we consider a state space where each link in the network is associated to two features: *i*) normalized link load, and *ii)* link weight. This is the same set of features that was used in [3], for a fair comparison between the two approaches.

Action Space: we consider a discrete action space of dimension $|E|$. An action consists in choosing a link. The chosen link has its associated weight incremented by one unit. Integer weights allow for the possibility of having multiple paths with the same total weight, in which flows will be equally split as per ECMP. Again, to keep the comparisons fair, this is the same action space that was used in [3].

Reward Function: we consider the immediate gain $R_t = O_{t+1} - O_t$, where O_t is the maximum normalized link load at iteration t. We also experimented with the absolute improvement $R_t = \max\{0,\ O^{\text{best}}_t - O_t)\}$, where O^{best}_t is the current best maximum link load found up to iteration t, but we observed little difference in performance.

Worst-case Complexity: each iteration of DeepLS requires recomputing the all-pairs shortest paths after the link weight update and the new OSPF routing. In the worst case, computing the all-pairs shortest paths in sparse graphs is $O(|V|^2 \log(|V|)$ with Johnson's algorithm. The forward pass in the DeepLS MLP neural network, considering it as a naive matrix multiplication, is $O(|E|F)$, where F is the number of edge features, and assuming the number of hidden neurons and hidden layers to be constant terms. Finally, computing the new OSPF routing with full traffic matrix is $O(|V|^3)$, since the number of hops in the routes are bounded by the total number of nodes. Assuming $F \ll |V|$, the worst-case time complexity of a single DeepLS iteration for OWS is $O(|V|^2 \log(|V|) + |V|^3) = O(|V|^3)$. The overall worst-case complexity for running DeepLS for N_{iter} iterations is therefore $O(N_{\text{iter}}|V|^3)$.

C. DeepLS for Routing and Wavelength Assignment

1) The Routing and Wavelength Assignment problem (RWA): RWA consists in assigning a route and a wavelength to a set of connection requests in an optical Wavelength Division Multiplexing (WDM) network. We assume transparent optical switching, hence wavelength continuity constraints must be enforced for each established lightpath. The objective is to maximize the number of established lightpaths. RWA can be expressed as an Integer Linear Programming (ILP) optimization problem as follows:

$$\max \sum_{p\in P} \sum_{w\in W} x^w_p \tag{8}$$

$$\sum_{p|l\in p} x^w_p \leq 1 \quad \forall l \in E, \forall w \in W \tag{9}$$

$$\sum_{p\in P_{(s,d)}} \sum_{w\in W} x^w_p \leq \rho_{(s,d)} \quad \forall s, d \in V, s \neq d \tag{10}$$

$$x^w_p \in \{0, 1\} \quad \forall p \in P, \forall w \in W \tag{11}$$

Where x_p^w indicates whether or not wavelength w is occupied in path p, P is a set of pre-computed paths, and $\rho_{(s,d)}$ is the number of connection requests between nodes s and d. Constraints (9) are wavelength continuity constraints, whereas constraints (10) ensure that the maximum number of wavelengths per link is not exceeded.

2) Formulating an LS for RWA as an MDP: we consider an LS algorithm where, at each iteration, the weight of one link is increased and a new feasible solution is computed with K-Shortest-Paths First-Fit (KSP-FF) on the weighted graph.

State Space: we consider a state space where each link in the network is associated to three features: *i)* normalized link load, *ii)* link weight, and *iii)* link betweenness centrality, defined as the fraction of shortest paths that traverse that edge. We empirically assessed that introducing this additional graph-level feature leads to improved performance.

Action Space: we consider a discrete action space of dimension $|E|$. An action consists in choosing a link. The chosen link has its associated weight incremented by one unit.

Reward Function: we consider the immediate gain $R_t = O_{t+1} - O_t$, where O_t is the blocking rate at iteration t.

Worst-case Complexity: each iteration of DeepLS requires computing the all-pairs K-shortest simple paths after the link weight update[2] and the routing and wavelength assignment with KSP-FF. Computing the K-shortest simple paths for a node pair in sparse graphs is $O(K|V|^2 \log(|V|)$ with Yen's algorithm. Therefore, computing the all-pairs K-shortest simple paths by iteratively calling Yen's algorithm is $O(K|V|^4 \log(|V|)$. As before, the forward pass in the DeepLS MLP neural network, being a matrix multiplication, is $O(|E|F)$. Finally, to evaluate the new solution value, we need to perform routing via KSP-FF. Determining the availability of a continuous wavelength over a path is $O(|V|W)$, where W is the number of wavelengths per link. Following up, checking for wavelength availability in each of the K-paths is $O(K|V|W)$. Routing all demands is therefore $O(K|V|WD)$, where D is the total number of demands. Assuming $F \ll |V|$, the worst-case time complexity of a single DeepLS iteration for RWA is $O(K|V|^4 \log(|V|) + K|V|WD)$. The overall worst-case complexity for running DeepLS for N_{iter} iterations is therefore $O(N_{\text{iter}} K(|V|^4 \log(|V|) + |V|WD))$

V. Illustrative Experimental Results

In our experiments, we used Proximal Policy Optimization (PPO) [25] for training the DRL agents, with the Stable Baselines implementation [32]. The Actor and Critic networks of PPO are implemented as the architecture illustrated in Fig. 2, where each MLP network has one single hidden layer with 16 neurons and Exponential Linear Unit (ELU) activation. In total, the two networks have $2 \cdot (m \cdot 16 + 16 + 16 + 1)$ trainable parameters, where m is the number of features associated to each decision variable. At inference time, only the policy network needs to be stored. In our problems, the number of parameters of the policy networks is 65 and 81 for OWS and RWA, respectively. Moreover, in all experiments we train on short episodes and evaluate on episodes at least one order of magnitude longer than training. Finally, being the environments CPU-bound, we leverage multiprocessing to run 6 environments in parallel for gathering experience. All of these tweaks allow to significantly reduce training times down to few minutes without sacrificing performance. All experiments were ran on a workstation with an Intel Core i5-8400 CPU (6 cores @2.80 GHz) and 32 GB RAM.

Our source code is publicly available at the following link: https://github.com/bonsai-lab-polimi/tnsm2023-deepls

A. OSPF Weight Setting problem

Training: we train only on the NSFNet topology with uniform traffic distribution. For fair comparison, we consider the same 100 training traffic matrices that were used in [3]. We train our model for 30000 episodes of length equal to N_{iter}=10. In each episode, a traffic matrix is randomly sampled from the 100 training ones. For PPO, we set the rollout buffer size to 128, and all other hyperparameters are left as their default values in the Stable-Baselines3 [32] implementation. The starting solution for DeepLS is computed as a standard OSPF routing with all links having unit weight. This means that our agent will be able to explore a small neighbourhood in which the total difference between the starting weights and the final weights can be at most equal to 10. On our hardware, training takes approximately 6 minutes. This is an improvement of more than two orders of magnitude compared to MARL-GNN, whose training times elapsed on average 11 hours on our hardware.

Evaluation: we evaluate on NSFNet, GBN and GEANT2 with both uniform and gravity-based traffic distributions, considering 100 test traffic matrices for each network-traffic combination. The traffic matrices used for testing are the same that were used in [3]. We therefore evaluate the generalization capabilities of DeepLS both on traffic distributions and network topologies unseen during training. We evaluate DeepLS on episodes of length N_{iter}=100 steps for NSFNet, N_{iter}=150 for GBN and N_{iter}=200 for GEANT2, for fair comparison to [3]. Results are averaged over five different training seeds.

Baselines: we compare DeepLS against the following:

- **Default OSPF**: OSPF routing resulting from having all link weights equal to one unit.
- **MARL-GNN** [3]: state-of-the-art GNN-based approach. We trained and evaluated MARL-GNN using the open-source code provided by the authors. For both training and inference we employed the same training configuration illustrated in the original paper. At inference time, we evaluate MARL-GNN in the same settings (i.e., topology and traffic distributions) in which it has been trained on. Results are averaged over five different training seeds.
- **LS-Greedy**: greedy local search heuristic which, at each timestep, selects the link that currently has the maximum link-load and increments its weight by one unit. For a fair comparison, LS-Greedy is run for a number of iteration N_{iter}=100, the same as DeepLS. The purpose of this baseline is to outrule the possibility that DeepLS learned a myopic greedy decision rule.

[2]As the link weights can only be increased, path recomputation can be performed only for the node pairs for which at least one of the K-shortest paths traverses the modified link. While not improving the worst-case asymptotic complexity, this leads to much shorter computational times in practice.

- **DEFO** [13]: state-of-the-art optimizer for traffic engineering, producing high-quality solutions. In particular, DEFO does not compute OSPF weights, but per-flow spitting ratios. These same solutions were considered as baseline in [3], which we extracted from their publicly available code repository. DEFO's bytecode is publicly available at [33], and the solutions reported are the best found in the fixed $30s$ time limit.
- **LP**: for each traffic matrix we compute the optimal solutions for the LP (4)-(7), which we report as a lower bound for the best possible solution that can be achieved.

Numerical Results: in Fig. 3 we show the CDF of the maximum link load achieved by DeepLS and by the baselines on the test traffic matrices for NSFNet, GBN and GEANT2. Generally speaking, we observe that DeepLS is able to generalize out-of-the-box to larger test set instances unseen at training time, outperforming MARL-GNN and attaining similar performance to DEFO in significantly shorter computational times. In particular, we underline that only DeepLS is evaluated on different network topologies and traffic distributions than training, whereas for MARL-GNN we kept the training and the evaluation settings the same.

Compared to LS-Greedy, DeepLS improves the maximum link load, on average, by 2.7% on NSFNet-Uniform, by 1.9% on NSFNet-Gravity, by 5.2% on GBN-Uniform, by 0.8% on GBN-Gravity, by 3.4% on GEANT2-Uniform, and by 3.0% on GEANT2-Gravity. As such, DeepLS consistently improves on the myopic LS-Greedy thanks to its lightweight policy.

Compared to the MARL-GNN [3], DeepLS improves the maximum link load, on average, by 5.6% on NSFNet-Uniform, by 1.7% on GBN-Uniform, by 4.8% on GBN-Gravity, by 2.6% on GEANT2-Uniform and by 1.5% on GEANT2-Gravity, while achieving slightly worse results (0.24% difference) only on NSFNet-Gravity. We remark that in NSFNet-Uniform, GEANT2-Uniform and GBN-Gravity the GNN-based approach is also outperformed by the myopic LS-Greedy, whereas our DeepLS approach outperforms on average LS-Greedy in all of the considered scenarios[3].

Furthermore, compared to the high-quality solutions computed by DEFO, DeepLS improves on average the maximum link load by 2.6% in GBN-Gravity, by 9.6% in GEANT2-Uniform and by 1.4% in GEANT2-Gravity, while worsening it by 5.1% in NSFNet-Uniform, 2.4% in NSFNet-Gravity, and by 2.7% in GBN-Gravity. Overall, DeepLS ties with DEFO, showing that it is on average capable to achieve solutions of similar quality than a state-of-the-art approach in highly diversified scenarios.

In terms of relative gap to the lower bound, DeepLS achieves on average a 10.2% relative gap, whereas the considered baselines achieve relative gaps equal to 10.6%, 13.3% and 14.24% for DEFO, LS-Greedy and MARL-GNN, respectively. Even though there is still a non-negligible optimality gap, on average DeepLS achieves slightly better performance than DEFO and significantly outperforms MARL-GNN. The optimality gap of DeepLS can be imputed to the constraint of equal flow split between shortest paths of equal cost. The LP can decide on arbitrary flow splits, hence providing only a lower bound on the optimal objective value for OWS.

We underline that OWS is a classical networking problem, extensively studied for several decades. We do not aim to outperform *all* state-of-the-art algorithm for OWS, but to demonstrate the potential of augmenting well-known algorithms with lightweight intelligence. In that regard, we have shown that a simple DRL-augmented Local Search becomes competitive with DEFO, a far more complex metaheuristic.

In Fig. 4 we illustrate the average running times per instance of the proposed DeepLS approach against LS-Greedy and the state-of-the-art MARL-GNN baseline. The times for LP are not shown, as the LP does not solve OWS, but provides only a theoretical lower bound on the solution value. We note that the times for DeepLS and LS-Greedy are nearly identical, showing that our lightweight neural architecture does not introduce additional overhead to the LS algorithm, while consistently improving on the solution quality. Compared to MARL-GNN, our lightweight MLP-only approach achieves on average an 83% speedup[4], while achieving solutions of better quality and with training times two order of magnitude smaller. A detailed comparison between the worst-case time complexities of DeepLS for OWS and the considered baselines is provided in Appendix A-A.

B. Routing and Wavelength Assignment problem

Training: we train on the NSFNet topology assuming 10 wavelengths per link and traffic matrices of 100 uniformly-distributed source-destination requests. We train our model for 10000 episodes of length equal to N_{iter}=10. For PPO, we set the rollout buffer size to 128, and all other hyperparameters are left as their default values in the Stable-Baselines3 [32] implementation. For each episode, a traffic matrix is randomly generated from a uniform distribution between all possible source-destination pairs. The starting solution for DeepLS is computed via KSP-FF with all links having unit weight. On our hardware, DeepLS takes approximately 10.6 minutes to train.

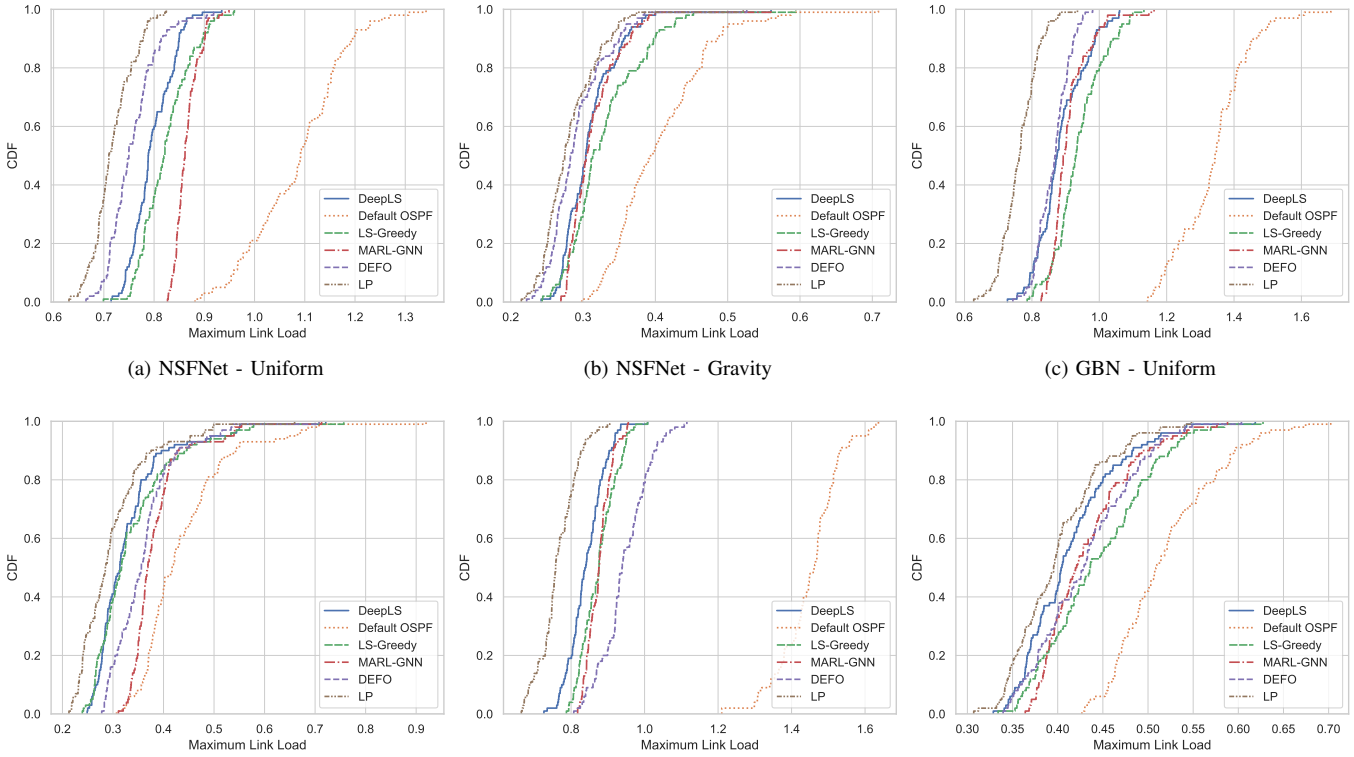
Evaluation: we evaluate DeepLS on NSFNet, GEANT2 and a synthetic 50-node Gabriel graph. Gabriel graphs have been observed, among different graph generators, to better capture the structure of physical layer network topologies [34]. We consider networks with 80 wavelengths per link and 800 uniformly distributed source-destination requests. We evaluate on episodes of length equal to N_{iter}=100 for each network topology. Results are averaged over five training seeds.

Baselines: we compare DeepLS against the following:

- **KSP-FF**: default K-Shortest-Paths First-Fit routing with links having equal unit weight. We consider K=3 shortest paths for each source-destination pair.

[3]This is somehow counterintuitive, as it appears that introducing a GNN module before the readout network is actually *worsening* the final performance. This can be imputed to a number of issues arising in GNNs such as oversmoothing [27] and oversquashing [28], which may hinder the overall final performance if not addressed properly.

[4]Authors of [3] used a slightly less time-efficient algorithm for recomputing the all-pairs shortest paths at each MARL-GNN iteration. With their algorithm, our relative speedup becomes 66% on average.



(a) NSFNet - Uniform (b) NSFNet - Gravity (c) GBN - Uniform

(d) GBN - Gravity (e) GEANT2 - Uniform (f) GEANT2 - Gravity

Fig. 3. CDFs of the maximum link load achieved by DeepLS and the considered baselines on 100 test traffic matrices.

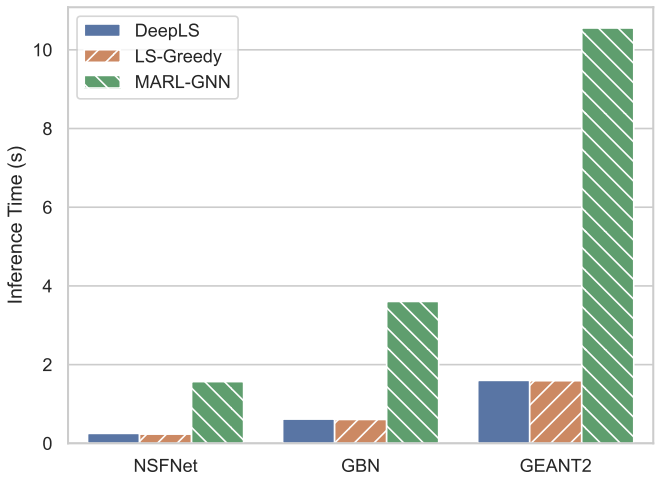
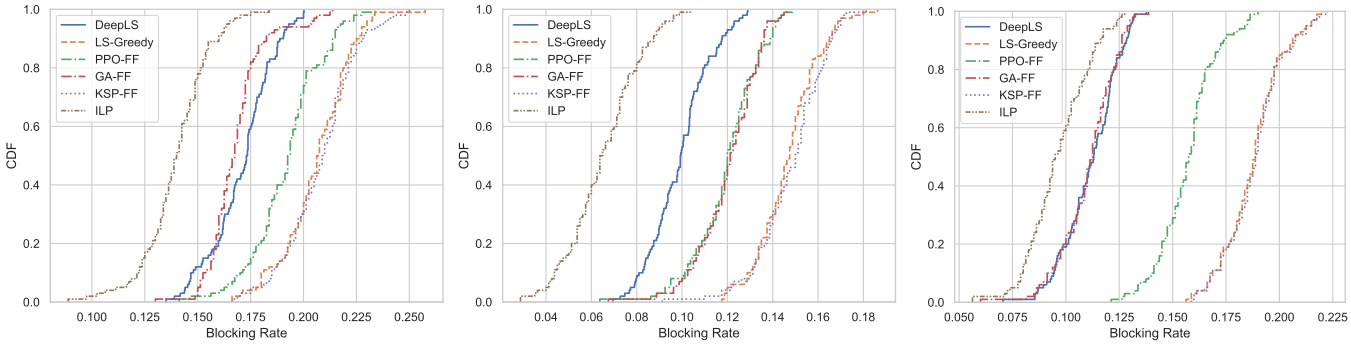


Fig. 4. Average computational times of DeepLS per OWS problem instance against LS-Greedy and the state-of-the-art MARL-GNN [3]. Computational times for DEFO [13] are fixed to 180s.

- **LS-Greedy**: greedy local search heuristic which, at each timestep, selects the link that currently has the maximum load and increments its weight by one unit. The link load is measured as the ratio between the number of occupied wavelengths and the number of available wavelengths. For a fair comparison, LS-Greedy is run for a number of iteration N_{iter}=100, the same as DeepLS.
- **PPO-FF** [6]: state-of-the-art DRL-based algorithm that processes the traffic matrix sequentially while querying a DRL agent for routing and admission control. The DRL agent either chooses one among the available K-shortest paths or for issuing a proactive rejection. In case of request admission and routing, wavelength assignment is performed by first-fit.
- **GA-FF** [35]: state-of-the-art Genetic Algorithm for solving RWA. In particular, GA-FF optimizes only the routing assuming the precomputation of K-shortest paths, and performs wavelength assignment by first-fit. We consider K=3 shortest paths for each source-destination pair. We consider a large population size of 1000 individual, and we run the algorithm until it reaches convergence, i.e., the objective function value is not improving after a large number of iterations.
- **ILP**: for each traffic matrix we compute the optimal solutions of the ILP (8)-(11), which we report as a lower bound for the best possible solution that can be achieved. We consider K=3 shortest paths for each source-destination pair.

Numerical results: in Fig. 5 the CDFs of the blocking rates achieved by DeepLS and the considered baselines are shown. Similarly to OWS, we observed that DeepLS generalizes out-of-the-box for larger problem instances unseen during training, achieving from similar to better solutions than GA-FF in significantly shorter computational times.

Compared to LS-Greedy, DeepLS achieves a relative improvement of 17% on NSFNet, 32% on GEANT2 and 40% on the 50-node Gabriel graph. Remarkably, LS-Greedy com-



(a) NSFNet (b) GEANT2 (c) Gabriel graph

Fig. 5. Empirical CDFs of the blocking rates achieved by DeepLS and the considered baselines on 100 test traffic matrices.

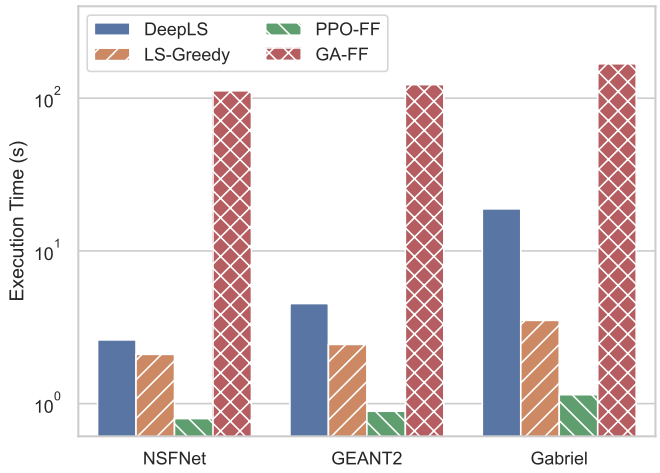


Fig. 6. Average computational times of DeepLS per RWA problem instance against a state-of-the-art Genetic Algorithm (GA) for RWA.

pletely fails to improve the starting KSP-FF solution. Empirically, we observed that LS-Greedy reaches stagnation almost immediately, repeatedly incrementing the weight of the same edge without improving the blocking rate. This illustrates that, unlike for the OWS problem, more sophisticated neighbourhood exploration strategies are required for achieving good quality solutions via Local Search.

Compared to PPO-FF, DeepLS achieves a relative improvement of 11% on NSFNet, 17% on GEANT2, and 29% on the 50-node Gabriel graph. These results illustrate that DeepLS brings a significant improvement in state-of-the-art DRL-based algorithms for solving static routing problems in optical networks. Moreover we underline that PPO-FF, while consistently outperforming KSP-FF, falls short against more complex traditional optimization algorithms such as GA-FF.

Finally, compared to GA-FF, DeepLS achieves solutions on average 17.6% better than GA-FF on GEANT2 and solutions of comparable quality on NSFNET and on the 50-node Grabriel graph (on average 1.6% and 0.8% worse, respectively). This is a remarkable result, showing that a weaker heuristic such as Local Search, enhanced by lightweight learning intelligence, can become competitive with respect to a handcrafted metaheuristic.

In Fig. 6 we illustrate the computational times of DeepLS compared to LS-Greedy, PPO-FF and GA-FF. The times for ILP are not shown: while it is possible to solve RWA on NSFNET in few seconds, computational times for GEANT2 and the 50-node Gabriel graph are out of scale. Compared to LS-Greedy, DeepLS is 48% slower: we empirically verified that this is because DeepLS triggers more K-shortest paths recomputations compared to LS-Greedy, leading to a more effective exploration of the solution space. Indeed, as shown before, LS-Greedy fails to improve the starting KSP-FF solution. PPO-FF is the fastest among all baselines since, by design, its runtime is only slightly slower than standard greedy heuristics such as KSP-FF. Unfortunately, as previously outlined, it often falls short in terms of solution quality compared to advanced metaheuristics. Compared to GA-FF, DeepLS achieves on average a 93% speedup, with computational times equal at most to $20s$ for the 50-node Gabriel Graph. Overall, DeepLS can compute solutions either on-par or better than a fine-tuned GA in significantly shorter computing times. A detailed comparison between the worst-case time complexities of DeepLS and the considered baselines is provided in Appendix A-B.

Again, DeepLS is able to generalize to larger problem instances on network topologies unseen during training. This allows us to keep training times to the minimum, as it is possible to train only on the least computationally intensive settings (i.e., smaller network topologies, smaller network capacity and lower total number of demands).

In terms of optimality gap with respect to the ILP, DeepLS achieves on average a 21.9% gap, whereas the baselines achieve gaps equal to 45.4% and 25.1% for LS-Greedy and GA-FF, respectively. The non-negligible optimality gap of DeepLS can be imputed to the limitations of a First-Fit wavelength assignment strategy. We underline that, for DeepLS, the smallest optimality gap of 13.1% was observed for the largest problem instances, i.e., on the 50-node Gabriel graph.

Again, RWA is a well-known problem in networking. Our aim is not to beat all state-of-the-art approaches for RWA, but to illustrate that simple algorithms enhanced with lightweight artificial intelligence can become more powerful than significantly more complex handcrafted algorithms. In

our case, DeepLS either outperforming or matching GA on the largest instances in far shorter computing times highlights the disruptive potential of ML alongside OR.

VI. Conclusion

In this paper, we proposed a lightweight DRL-augmented Local Search algorithm (DeepLS), and we applied it to the OSPF Weight Setting problem and the Routing and Wavelength Assignment problem. Leveraging a permutation-equivariant neural network operating on sets of decision variables, DeepLS can evaluate problem instances of arbitrary size, showing remarkable generalization capabilities to network topologies, traffic distributions, network capacities and network loads unseen during training. DeepLS, composed by less than a hundred parameters, outperforms state-of-the-art DRL-based approaches in literature and achieves competitive performance compared to state-of-the-art metaheuristics, while being on average faster than both. Overall, our results show that incorporating lightweight learning intelligence in existing methodologies can be the cornerstone for designing powerful and scalable network-optimization algorithms. As the proposed methodology is general, future research directions include applying DeepLS to more complex neighbourhood definitions (e.g., rerouting, ejection chains), more complex network-optimization problems (e.g., Routing, Modulation format and Spectrum Assignment in Elastic Optical Networks, optimization of Service Function Chaining), and more sophisticated metaheuristic frameworks (e.g., Iterated Local Search, Variable Neighbourhood Search, Ruin&Recreate).

Acknowledgement

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

Appendix A
Worst-Case Complexity Analysis

In this Appendix, we compare the worst-case complexities of the considered baselines against DeepLS for both the OSPF Weight Setting problem and the Routing and Wavelength Assignment problem.

A. OSPF Weight Setting problem

We previously demonstrated that the worst-case time complexity of DeepLS for OWS is $O(N_{\text{iter}}|V|^3)$. The worst-case time complexities of the considered baselines are as follows:

- **Default OSPF**: requires computing the all-pairs shortest paths and distributing the demands. Hence, as shown for a single iteration of DeepLS, the complexity is $O(V^3)$.
- **MARL-GNN** [3]: requires a forward pass on the line graph of the network topology. The time complexity of the forward pass is therefore proportional to the number of edges in the line graph of the network topology, i.e., $O(L|V||E|)$, where L is the number of GNN layers. As the algorithm requires at each iteration to recompute all-pairs' shortest paths and to distribute the demands, the overall time complexity is $O(N_{\text{iter}}(|V|^3 + L|V||E|))$.
- **LS-Greedy**: equal to DeepLS, as LS-Greedy misses only the neural network policy, whose computational complexity can be elided.
- **DEFO** [13]: with reference to [36], DEFO alternates between destruction and recreation procedures for generating an optimal forwarding plane configuration. The time complexity of recreation is $O(N_{\text{iter}}|E||V|^2 \log |V|))$, as it requires repeated sorting of the nodes and propagation of the link load constraints. The time complexity of destruction, compared to recreation, can be elided.

We underline that asymptotically running DeepLS is as complex as running the Default OSPF heuristic N_{iter} times. Indeed, the complexity is upper-bounded by the evaluation of the objective function value and the link loads. Finally, while MARL-GNN and DeepLS show similar worst-case time complexities, we illustrated in Fig. 4 that DeepLS scales much better with instance size thanks to its extremely lightweight neural network architecture.

B. Routing and Wavelength Assignment problem

We previously demonstrated that the the worst-case time complexity of DeepLS for RWA is $O(N_{\text{iter}}K(|V|^4 \log(|V|) + |V|WD))$. The worst-case time complexities of the considered baselines are as follows:

- **KSP-FF**: requires computing the all-pairs' K-shortest paths and to distribute the demands. Therefore, the worst-case complexity is $O(K|V|^4 \log(|V|) + K|V|WD)$.
- **LS-Greedy**: equal to DeepLS, as LS-Greedy misses only the neural network policy, whose computational complexity can be elided.
- **PPO-FF** [6]: requires computing the all-pairs' K-shortest paths and querying the DRL agent for each connection request in the traffic matrix. As the state space of PPO-FF is $O(|V|)$, the computational complexity of routing the full traffic matrix is $O(K|V|^2WD$. Therefore, the computational complexity of PPO-FF is $O(K(|V|^4 \log(|V|) + |V|^2WD))$.
- **GA-FF** [35]: the algorithm optimizes the ordering of demands and the chosen routing, whereas wavelength assignment is performed with a first-fit rule. As such, the worst-case complexity of the algorithm is bounded by the fitness function calculation, i.e., the evaluation of the objective function value given a request ordering and a choice of routing for every request. Assuming that the all-pairs' K-shortest paths have been precomputed, the overall time complexity is $O(N_{\text{iter}}K|V|WD \cdot \text{Pop\_Size})$, where Pop_Size is the population size (i.e., the number of candidate solutions to be kept at each iteration of GA-FF). In practice, it holds that Pop_Size $= O(KD)$, as a common rule of thumb is to set the population size proportional to the number of variables (i.e., genes) in a candidate solution, which in our cases are one per demand. Hence, the overall time complexity is $O(N_{\text{iter}}K^2|V|WD^2)$.

Similarly to what we observed for OWS, running DeepLS is as time-consuming as running KSP-FF N_{iter} times. In this regard, while PPO-FF scales significantly better than DeepLS, as it does not have to recompute the K-shortest-paths, but significantly underperforms in terms of solution quality. Finally, we underline that while the time-complexities for DeepLS and GA-FF may be difficult to compare in the current form, it is reasonable to assume that $O(D) = O(|V|^2)$ (e.g., considering a full-mesh traffic matrix). Hence, the time-complexities of DeepLS and GA-FF further simplify to $O(KN_{\text{iter}}(|V|^4 \log(|V|) + W|V|^3))$ and $O(N_{\text{iter}}K^2W|V|^5)$, respectively. Therefore, DeepLS not only provides similar to better solutions than GA-FF, but also scales better with respect to the instance size, as we empirically demonstrated in Fig. 5 and Fig. 6.

REFERENCES

[1] D. Andreoletti, S. Troia, F. Musumeci, S. Giordano, G. Maier, and M. Tornatore, "Network traffic prediction based on diffusion convolutional recurrent neural networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 246–251.

[2] C. Natalino, C. Manso, L. Gifre, R. M. noz, R. Vilalta, M. Furdek, and P. Monti, "Microservice-based unsupervised anomaly detection loop for optical networks," in *Optical Fiber Communication Conference (OFC) 2022*. Optica Publishing Group, 2022, pp. 1–3.

[3] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Is machine learning ready for traffic engineering optimization?" in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, 2021, pp. 1–11.

[4] D. Aureli, A. Cianfrani, M. Listanti, and M. Polverini, "Intelligent link load control in a segment routing network via deep reinforcement learning," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2022, pp. 32–39.

[5] X. Chen, J. Guo, Z. Zhu, R. Proietti, A. Castro, and S. J. B. Yoo, "Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, 2018, pp. 1–3.

[6] N. Di Cicco, E. F. Mercan, O. Karandin, O. Ayoub, S. Troia, F. Musumeci, and M. Tornatore, "On deep reinforcement learning for static routing and wavelength assignment," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 28, no. 4, pp. 1–12, 2022.

[7] A. H. C. Correia, D. E. Worrall, and R. Bondesan, "Neural simulated annealing," 2022. [Online]. Available: https://arxiv.org/abs/2203.02201

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016.

[9] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, 2015.

[10] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, 2019.

[11] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[12] "EURO Meets NeurIPS 2022," https://euro-neurips-vrp-2022.challenges.ortec.com/, 2022, [Online; accessed 11-December-2022].

[13] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, p. 15–28.

[14] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, p. 185–191.

[15] O. Hope and E. Yoneki, "Gddr: Gnn-based data-driven routing," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 517–527.

[16] M. R. Raza, C. Natalino, P. Öhlen, L. Wosinska, and P. Monti, "Reinforcement learning for slicing in a 5g flexible ran," *Journal of Lightwave Technology*, pp. 1–12, 06 2019.

[17] F. Kosanoglu, M. Atmis, and H. Turan, "A deep reinforcement learning assisted simulated annealing algorithm for a maintenance planning problem," *Annals of Operations Research*, 03 2022.

[18] M. Alicastro, D. Ferone, P. Festa, S. Fugaro, and T. Pastore, "A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems," *Computers & Operations Research*, vol. 131, 2021.

[19] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[21] C. Nota and P. S. Thomas, "Is the policy gradient a gradient?" 2019. [Online]. Available: https://arxiv.org/abs/1906.07073

[22] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[26] O. Vinyals, I. Babuschkin, W. M. Czarnecki *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, p. 350–354, 2019.

[27] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3438–3445, Apr. 2020.

[28] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=i80OPhOCVH2

[29] S. Zhang, Y. Liu, Y. Sun, and N. Shah, "Graph-less neural networks: Teaching old MLPs new tricks via distillation," in *International Conference on Learning Representations*, 2022.

[30] P. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf

[31] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, pp. 466–474.

[32] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[33] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "DEFO website," https://sites.uclouvain.be/defo/, 2015.

[34] E. K. Çetinkaya, M. J. Alenazi, Y. Cheng, A. M. Peck, and J. P. Sterbenz, "On the fitness of geographic graph generators for modelling physical level topologies," in *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2013, pp. 38–45.

[35] O. Karandin, F. Musumeci, O. Ayoub, A. Ferrari, Y. Pointurier, and M. Tornatore, "Quantifying resource savings from low-margin design in optical networks with probabilistic constellation shaping," in *2021 European Conference on Optical Communication (ECOC)*, 2021, pp. 1–4.

[36] R. Hartert, P. Schaus, S. Vissicchio, and O. Bonaventure, "Solving segment routing problems with hybrid constraint programming techniques," in *Principles and Practice of Constraint Programming*, G. Pesant, Ed. Cham: Springer International Publishing, 2015, pp. 592–608.