Self-Renewal Machine Learning Approach for Fast Wireless Network Optimization

[1]Oluwaseun T. Ajayi, [2]Xianghui Cao, [3]Hangguan Shan, and [1]Yu Cheng
[1]Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, 60616, USA
[2]School of Automation, Southeast University, Nanjing 210096, China
[3]College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China
Email: oajayi6@hawk.iit.edu; xhcao@seu.edu.cn; hshan@zju.edu.cn; cheng@iit.edu

***Abstract*—The throughput maximization in multi-hop wireless networks is largely limited by interference due to the reuse of the channel resources. Although machine learning (ML) can accelerate the optimization of wireless network capacity, however, the existing system can become limited because of insufficient knowledge from available data. We propose a self-renewal ML (SRML) method that incrementally improves the throughput maximization of future optimization instances through the design of a data selection algorithm for scheduling structure classification and application identification model retraining. With one round of implementation, the SRML method outperforms the fixed ML (FML) method, random (RAND) method and Greedy Heuristic method in the multi-commodity flow deployment setting with an average achievable throughput of 100% for small flows and at least 79% for large flows, relative to the delayed column generation (DCG) benchmark algorithm, while reducing the computational complexity and achieving a high solution efficiency. By leveraging the transfer learning of parameters during self-renewal, the computational cost of model training is reduced by at least 71.09%.**

***Index Terms*—Capacity optimization, machine learning, scheduling, self-renewal, wireless multi-hop**

I. Introduction

The proliferation of mobile devices and internet applications has paved way for big data analytics in wireless networking (e.g cellular networks, mobile ad-hoc networks, satellite communication networks, integrated access and backhaul networks, etc.), thus requiring the intelligent exploitation of such data for improved network performance across the application layer, network layer and link layer [1]. Fundamentally, the time and space resources in wireless networks are so limited that they need to be optimally scheduled to satisfy both network quality and user traffic requirements. However, resource allocation is considered as a complex optimization problem which can be modeled as an independent set problem (like the graph coloring of links), and is NP-hard [2]–[4]. The independent set problem is, finding the set of wireless links that can be scheduled for simultaneous transmission without interference or communication collision. In a time-slotted wireless network, different independent sets (ISs) grab the channel for transmission in non-overlapping timeslots to complete a network flow demand [2].

Most of the efforts to tackle the complex resource allocation problem in wireless networks are towards the development of approximation algorithms [5]–[7] which are mathematically intense and computationally expensive [4], and they do not always guarantee global optimality. Heuristic pruning algorithms such as in the cooperative communication aware link scheduling [8] in vehicular ad-hoc networks (VANETs) have been proposed, where the throughput maximization problem is NP-complete and can be solved near-optimally by linear programming. The extended link-band pairs in [8] makes the solution space larger where a relay node needs to be selected as either a cooperative relay node or a multi-hop relay node. Resource scheduling in multidimensional resource space is, thus, challenging to solve with mathematical programming [8], [9]. However, the delayed column generation (DCG) approximation algorithm has been found to serve as a feasible benchmark for achieving near-global optimality for the network flow problem which is solved by finding the optimal schedulable ISs [4]. While mathematical algorithms seem to be a viable approach to resource scheduling problems, they only solve the optimization instances in a case-by-case manner without any knowledge retention, thus, not benefiting a typical wireless network (e.g wireless mesh networks, VANETs, and wireless sensor networks) where a previously solved optimization instance can be similar to a future optimization instance.

To address the limitations with the use of approximation algorithms, machine learning (ML) is being explored to facilitate resource allocation decisions based on the experience from historical optimization instances [2], [10]. While the early works leveraging ML focused on the single-hop networks [11], the application of ML to multi-hop wireless networks has gained recent attention [2], [10], [12], [13]. The work in [12] proposed an ML framework to reduce the problem scale by pruning off unimportant links which are not part of the solution to the network flow problem; this was extended to the topology-aware setting in [13] by leveraging the graph embedding technique. While [12], [13] consider an indirect ML approach for solving the optimization problem, the recent work in [2] proposed a self-supervised learning framework that partially explores the ISs of historical optimization instances to accelerate the optimization of future instances. However, the existing method in [2] can become limited, as it assumes an isolated learning or fixed ML (FML) framework (i.e learning is fixed and parameters cannot be updated) in which the trained ML model does not generalize well on larger network flows that exceed the ones seen during training. For example, if the

model is trained with instances from 1 flow to 5 flows, the performance on 10 flows degrades, thus yielding a maximum achievable throughput that is sub-optimal as evidenced by the numerical experiments in [2].

Since it is difficult to obtain sufficient data that covers all the possible commodity flow deployments within a wireless network, we opine that an ML model should be intelligent to continuously learn new knowledge, i.e when the model's performance degrades on a set of new optimization instances that are not part of the base training data, it should update its knowledge through an offline self-renewal procedure. This will facilitate the solution of future optimization instances whose complexity has been learned from the self-renewal process. Although the complex optimization problem can be cast as a Markov decision process (MDP) and solved using the deep reinforcement learning (DRL) technique, however, the iterative operations based on the adjustment of action policies makes this approach inefficient and time consuming to solve for larger networks with disparate flow deployments [10]. Hence, in this paper, we propose an ML framework with self-renewal capability to assist the optimization of network capacity.

The self-renewal ML approach can also be referred to as "continual learning" or "lifelong machine learning" in the literature [14]–[16]. However, in the context of wireless network optimization, we adopt the "self-renewal" term since the learning of resource scheduling solutions can be finite rather than lifelong, when sufficient knowledge has been learned. Our contributions are summarized as follows:

- We collected historical optimization instances solved using the DCG benchmark algorithm as base training data. We implemented the Dunn's index algorithm [17] to find the number of clusters and leveraged a clustering algorithm to group the *scheduling structures* (referring to the scheduled ISs) of the historical optimization instances into appropriate number of scheduling classes.
- We trained an application identification (AID) model using the application-level information (e.g topology, commodity flow deployment, link capacity, user traffic constraints, etc.) as input and the clustering algorithm's output as labels. Each label corresponds to a class of schedulable ISs which can be used to retrieve the ISs to solve an optimization instance through one-round of linear programming (LP).
- We designed an intelligence criterion α for the AID model, to facilitate the future optimization of complex flow demands by relearning the scheduling structures. This requires re-clustering (i.e either to add a new cluster or expand existing ones) and retraining the AID model.
- The AID model retraining follows the transfer learning technique where the pre-trained weights of the base model are used to instantiate training to speed up learning.
- We introduce the solution efficiency (SE) metric to evaluate the AID model's performance with respect to throughput maximization and number of training samples.

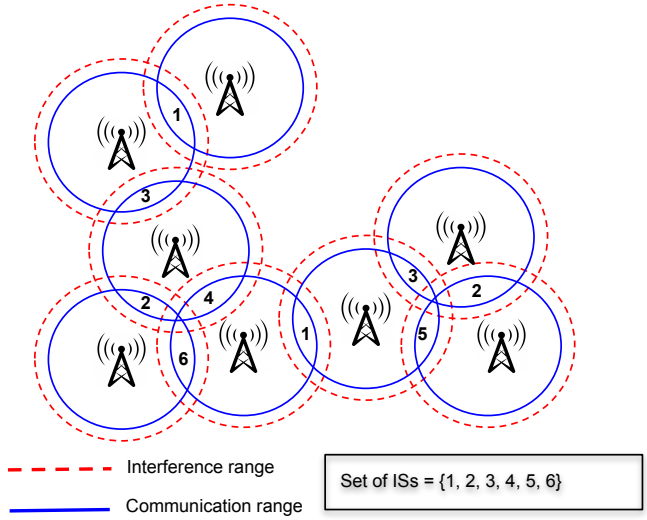

Fig. 1: System model illustration.

The proposed self-renewal ML (SRML) method is applied to a small-scale network and a large-scale network of 30 nodes and 100 nodes respectively. Through experiments, our method achieves a maximum achievable throughput of up to 100% relative to the DCG benchmark algorithm while reducing the computational complexity by up to 97%. Further to that, we show the strength of our method as against the FML approach in [2] using the SE metric and obtained up to 6.40% improvement. The remainder of this work is organized as follows. Section II presents the system model and problem formulation. Section III presents the proposed SRML framework, while Section IV presents the numerical experiments and results. We give the related works in Section V, and then the conclusion in Section VI. In this paper, vectors and matrices are denoted by lowercase and uppercase boldface letters respectively. Table I lists the main notations used in this paper.

TABLE I: Main notations.

Notation	Definition
$\mathcal{N}$	set of nodes
$\mathcal{E}$	set of wireless links
$e_{m,n}$	communication link from node m to node n
$e^{cap}_{m,n}$	Shannon capacity achievable on link $e_{m,n}$
$\mathcal{N}^+_n$	set of out-neighbor nodes of n
$\mathcal{N}^-_n$	set of in-neighbor nodes of n
$\mathcal{A}$	set of all schedulable ISs
$\mathcal{A}^*$	set of critical ISs
γ_a	transmission time of a scheduled IS $a \in \mathcal{A}$
p_d	total throughput of flow d
w_d	weight of a commodity flow d
$f_d(m,n)$	amount of flow d on link $e_{m,n}$
(s_d, t_d)	source-destination tuple of flow d
τ	model update trigger

II. System Model and Problem Formulation

We consider a generic multi-hop single-radio single-channel wireless network which is modeled by a graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ with $\mathcal{N}$ and $\mathcal{E}$ denoting the set of nodes and physical links

respectively. A direct communication link $e_{m,n} \in \mathcal{E}$ exists if the receiver of node n is within the communication range of node m. The maximum achievable capacity $e^{cap}_{m,n}$ is derived from the Shannon capacity formula as:

$$e^{cap}_{m,n} = Blog_2(1 + \psi_{e_{m,n}}) \tag{1}$$

where B and $\psi_{e_{m,n}}$ denote the channel bandwidth and signal-to-interference plus noise ratio (SINR) respectively. We adopt the *protocol interference* model [18] for the scheduling of links, which stipulates that for a successful transmission from node m to node n, no other node must transmit within the interference range of node n, and also that links sharing the same node cannot be activated concurrently due to the half-duplex property of nodes. We use Figure 1 to illustrate the system model of a wireless network with 8 nodes (e.g base stations or access points) and 9 links. The wireless links[1] are represented by the overlapping communication range of the nodes, and for a successful transmission at a certain moment, only a set or group of interference-free links are activated simultaneously. In Figure 1, we show an illustration of six sets[2] of links that can be activated independently. Each set contains at least a link that can be activated during a transmission period or time-slot. For example, sets 1, 2 and 3, contain two activated links each, while sets 4, 5 and 6 contain one activated link each.

The throughput maximization problem thus maps to the scheduling of links and time allocation to satisfy commodity flow requirements by formulating a multi-commodity flow (MCF) problem which is to be solved by LP. We denote the set of commodity flow demands by $\mathcal{D}$, with each flow $d \in \mathcal{D}$ representing a source-destination tuple as $d = (s_d, t_d)$. In an interference-prone wireless network, only non-interfering links can be activated for simultaneous transmission, forming an independent set (IS); this corresponds to a set of vertices without connected arcs in the conflict graph. Several ISs within a set $\mathcal{A}$ need to be scheduled in a non-overlapping time-sharing manner (e.g TDMA) to satisfy the flow demands. Note that the number of ISs in $\mathcal{A}$ can be exponentially large, in the order of $2^{|\mathcal{E}|}$. For each IS $a \in \mathcal{A}$, the allocated transmission time is denoted by γ_a, and the effective capacity of a link is:

$$e^a_{m,n} = \begin{cases} e^{cap}_{m,n} & \text{if } e_{m,n} \text{ is active in IS } a \\ 0 & \text{otherwise} \end{cases}$$

Also, we denote by $f_d(m,n)$ the amount of flow d carried on link $e_{m,n}$. For a commodity flow d, the total achievable throughput p_d is the net amount of flow out of the source s_d, i.e

$$p_d = \sum_{n \in \mathcal{N}^+_{s_d}} f_d(\mathrm{s}_d, n) \tag{2}$$

[1]For the establishing of links, we assume the use of an omnidirectional (or isotropic) antenna with power radiating equally in all directions. Nodes within each other's communication range, thus forms a wireless link.

[2]A set or group of activated links is also referred to as an independent set.

where $\mathcal{N}^+_{\mathrm{s}_d}$ denotes the set of out-neighbor nodes of s_d. We define the following constraints for the formulated MCF problem (denoted by $\mathbb{P}$) where $f_d(m,n)$ and γ_a are the decision variables:

1) Flow conservation constraint: The total amount of incoming traffic flows to a node that is neither the source nor the destination must be equal to the total amount of outgoing traffic flows from that node. This implicitly determines the routing of each flow demand [2].

$$\sum_{m \in \mathcal{N}^-_n} f_d(m,n) = \sum_{m' \in \mathcal{N}^+_n} f_d(n,m'), \quad \forall n \neq \mathrm{s}_d, \mathrm{t}_d; \forall d. \tag{3}$$

2) Link capacity constraint: The total amount of flows allocated to a link must not exceed its maximum capacity over all the scheduled ISs.

$$\sum_{d \in \mathcal{D}} f_d(m,n) \leq \sum_{a \in \mathcal{A}} \gamma_a e^a_{m,n}, \quad \forall e_{m,n} \in \mathcal{E} \tag{4}$$

3) Scheduling constraint: The amount of time allocated to all the ISs must sum to 1 (in a normalized fashion).

$$\sum_{a \in \mathcal{A}} \gamma_a = 1 \tag{5}$$

The objective of $\mathbb{P}$ is to maximize the weighted sum of achievable network capacity (i.e throughput), defined as:

$$\begin{aligned} \underset{\{f_d(m,n)\},\{\gamma_a\}}{\text{Maximize}} \quad & \sum_{d \in \mathcal{D}} w_d p_d \\ \text{s.t.} \quad & \text{constraints } (3), (4), (5), \\ & f_d(m,n) \geq 0, \quad \forall e_{m,n} \in \mathcal{E}, \forall d \\ & \gamma_a \geq 0, \quad \forall a \in \mathcal{A} \end{aligned} \tag{6}$$

Each weight w_d represents user service-level priority. No doubt (6) can be solved using the DCG algorithm, which works by iteratively solving a restricted master problem and sub-problem. The sub-problem involves searching and adding a new entering column (i.e an IS) to the master problem to improve the achievable throughput. Although the DCG algorithm is able to converge to the optimum value, however, solving the sub-problem is NP-hard. We use the DCG algorithm to generate the training cases, and it also serves as a comparative benchmark for the proposed SRML method.

III. Proposed Self-Renewal ML Framework for Wireless Network Optimization

The network capacity region, termed as the convex hull to an optimization problem, can be obtained by different sets of ISs, however, there is always a subset of these sets of ISs that achieves the optimal solution. This subset is called the critical ISs (CISs) denoted by $\mathcal{A}^*$, as it guarantees the maximum achievable throughput. Note that the CISs is a subset of the set of the exponential ISs, i.e $\mathcal{A}^* \subset \mathcal{A}$ and $|\mathcal{A}^*| << |\mathcal{A}|$. The DCG approximation algorithm solves the optimization problem to find the CISs that maximizes the

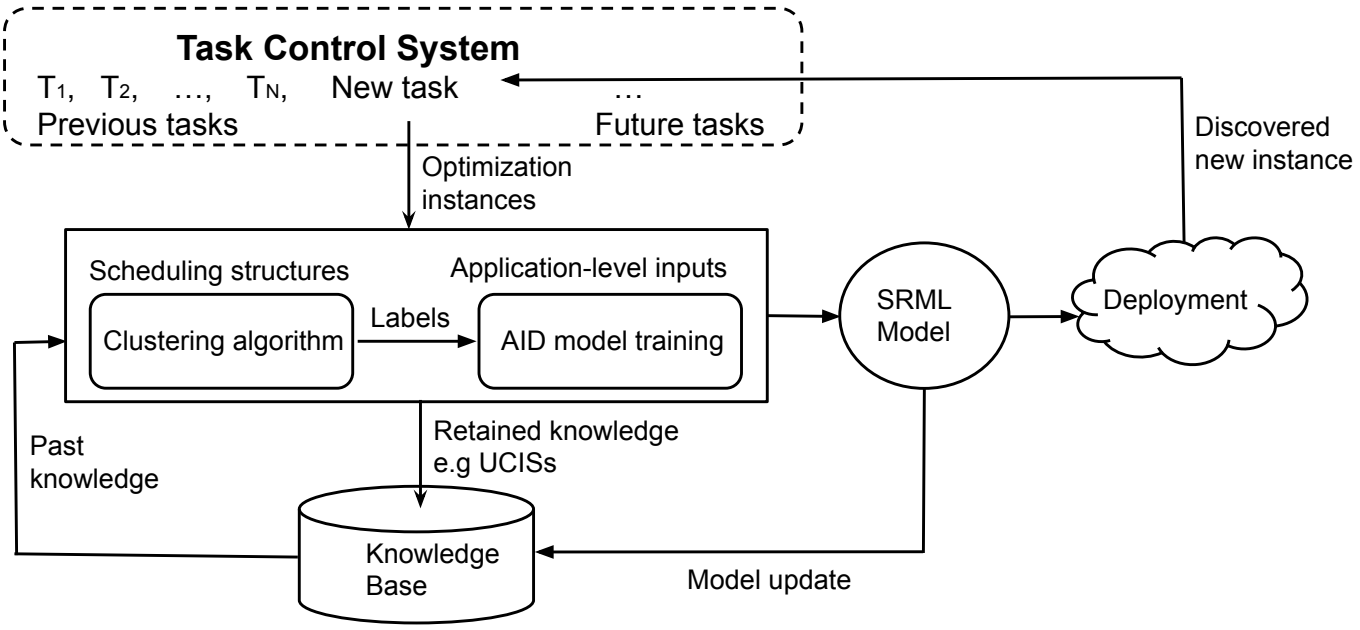

Fig. 2: The self-renewal machine learning framework.

network throughput. Beyond the computational complexity of the DCG algorithm, there is also no knowledge retention of solved optimization instances. We intuit that the CISs of historical optimization instances can be exploited smartly to solve a future optimization instance using well-designed ML techniques. However, in the scarce data regime (which is the case in wireless network optimization), the capability of ML becomes limited, except if it is based on lifelong learning[3] [15], i.e having the capability to improve on existing knowledge as new data are observed.

Designing an SRML framework in the wireless network context is not trivial as is the case in other domains for the following reasons: 1) learning every newly observed data will resort to the rote memorization of ML, and 2) identifying newly observed data that will improve ML capability requires the careful design of a data selection algorithm. In this paper, we address these two challenges by first designing a two-stage ML technique [2] and then implement the SRML procedure to improve the solution quality of future optimization instances. The architecture of the self-renewal ML framework is shown in Figure 2, which is an extension to the self-supervised learning framework in [2]. The main components are:

1) Task control system: It receives, processes and manages the tasks that arrive in the system. When a new task (i.e batch of optimization instances) arrives, it does the offline computation of the DCG solution and sends it to the learner where scheduling structure classification and AID model training are performed.

2) SRML model: The learner completes the learning task, and outputs the SRML model for deployment. The SRML model is the trained and updated model.

3) Knowledge base: It stores all the previously learned knowledge from the model (e.g schedulable ISs, model parameters, etc.) and gets the newest knowledge after model updating.

From the architecture, the learner performs two-stages of learning, i.e scheduling structure classification and AID model training.

A. Scheduling Structure Classification

The scheduling structure of an optimization instance refers to the scheduled CISs used to obtain the solution to $\mathbb{P}$ in (6). For the optimization of a new instance, the scheduling structure of historical optimization instances, called the union of CISs (UCISs) can be utilized to approximate the solution. The UCISs are obtained by storing the CISs of instances that belong to the same class. Such class is derived from clustering, by learning the similarity between the scheduling structures of historical computed instances.

The scheduling structure of an optimization instance is represented by an $|\mathcal{E}|$-dimensional vector $\mathbf{x}$ (corresponding to the number of links), and computed by the column-wise averaging of the scheduled CISs where the e-th element in an IS refers to the activation state of a link (either 1 or 0). The Kmeans++ algorithm is then used to cluster similar scheduling structures. Since this is an unsupervised learning task, there is no prior knowledge on what number of clusters to group the data into. The popular Dunn's index algorithm [17], which computes the ratio between the minimum distance between two clusters and the size of the largest cluster, is used to obtain the optimal K value as an input parameter to the Kmeans++ algorithm, as follows:

[3]Lifelong learning is an ML paradigm in which knowledge learned from previous tasks is used to help future learning. This way, a model can learn to solve unfamiliar problems.

$$K^* = \arg\max_K \left(\frac{\min\limits_{1\leq i\leq K}\left\{\min\limits_{1\leq j\leq K}\{\|\mu_i - \mu_j\|\}\right\}}{\max\limits_{1\leq k\leq K}\{|C_k|\}} \right) \quad (7)$$

where μ_i and μ_j denote the centroids of clusters C_i and C_j respectively, and $|.|$ denotes the number of data points in a cluster. The clustering algorithm computes several iterations to minimize the total sum of squared distance between the data points and the K centroids as follows:

$$\underset{\mu_1,\cdots,\mu_K\in C}{\text{minimize}} \quad \sum_{k=1}^{K}\sum_{\mathbf{x}\in C_k} \|\mathbf{x} - \mu_k\|^2 \quad (8)$$

As a preprocessing stage, to avoid the effect of the "curse of dimensionality" during clustering, we used the principal components analysis (PCA) algorithm [19] to reduce the dimensions of the scheduling structure vector $\mathbf{x}$ before performing clustering with the Kmeans++ algorithm. The explained variance ratio (EVR) in the PCA serves as a guide to determine the number of principal components (PCs) to use in preserving the salient features in the data. In the experiment, the PCs that achieve an EVR of at least 0.7 is used for clustering.

B. Application Identification Model

From the clustering of the scheduling structures, the class labels $\{c_1, \cdots, c_K\}$ are obtained and are mapped to the instance vectors $\mathbf{v}^{(i)} \in \mathcal{V}$ to train an AID model in a supervised learning manner, where $\mathbf{v}^{(i)}$ is an application-level input vector (containing e.g topology, link capacities, flow deployment, flow weights, etc.) of instance i. Specifically, we encode the application-level input by defining an adjacency matrix $\mathbf{V} \in \mathbb{R}^{|\mathcal{N}|\times|\mathcal{N}|}$ and a flow weight vector $\mathbf{w} = (w_1, \cdots, w_{|\mathcal{D}|})$. An instance vector $\mathbf{v}^{(i)}$ is thus defined as:

$$\mathbf{v}^{(i)} = Concat\left(Vec(\mathbf{V}), \mathbf{w}\right) \quad (9)$$

The $Concat$ and Vec operations are for concatenation and matrix-to-vector transformation respectively. In our experiment, the dimension of $\mathbf{v}^{(i)}$ is fixed as we assume that no node leaves the network and that no new nodes join the network. For each network size of 30 nodes and 100 nodes in our experiment, the dimensions are different. Therefore, we trained two AID models, each, for the small-scale and large-scale networks (also note that the clustering of scheduling structures is performed separately for these network sizes). The AID model is a fully-connected neural network $f(\theta, \mathbf{V})$ which is designed to learn the mapping relationship between the application-level input of historical optimization instances and their scheduling structures, i.e $\mathbf{v}^{(i)} \rightarrow c_i$, where c_i corresponds to the UCISs of the instances in class c_i. The *categorical cross-entropy* loss function is used to compute model performance and for backpropagation.

The list of hyperparameters for training the AID models are given in Table II, with $\mathbf{W}_{ij}$ denoting the weight matrix of layer i in model j, where $j = 1$ for the 30 nodes network and $j = 2$ for the 100 nodes network. We also use a "," to separate the hyperparameter values for the 30 nodes network and 100 nodes network respectively.

TABLE II: Neural network parameters and hyperparameters

Name	Value
Epochs	27, 193
Batch size	500
Layers	5, 4
Hidden layer function	ReLU
Output layer function	Softmax
Optimizer	Adam
Learning rate	0.0006, 0.0001
Dropout	None, 0.25
Regularization	Early stopping
$\mathbf{W_{11}}, \mathbf{W_{12}}$	896×910, 448×10010
$\mathbf{W_{21}}, \mathbf{W_{22}}$	608×896, 864×448
$\mathbf{W_{31}}, \mathbf{W_{32}}$	640×608, 832×864
$\mathbf{W_{41}}, \mathbf{W_{42}}$	960×640, 1000×832
$\mathbf{W_{51}}$	1000×960

C. Self-Renewal Procedure

The self-renewal of the AID model stems from the need to incrementally improve the throughput of future instances by relearning the scheduling structures and learning the UCISs mapping to new application-level inputs that were not part of the base training data. This is similar to the continuous learning or lifelong learning paradigm of supervised ML [14], [15], except that instead of using all the newly observed instances $\mathcal{B}$, we only select a subset of them, i.e $B' \subset \mathcal{B}$ to improve the throughput of future instances, thus slightly increasing the size of the dataset and reducing the number of iterations required for model updating. To update the model implies retraining it with previous data and new data. We explain and describe the data selection and model retraining procedure for the SRML method in Algorithm 1 and Algorithm 2 respectively.

The AID model retraining can be time-based or need-based, with the former occuring within a defined period (e.g weekly, monthly, or yearly), and the latter defined by the availability of new data or performance monitoring. We use the latter strategy to update the AID model by selecting a data batch B' from the test (or inference) set $\mathcal{B}$, based on an intelligence criterion $\alpha \in (0, 1]$ that defines the minimum confidence score of $f(\theta, \mathbf{v}^{(i)})$ as shown in Algorithm 1.

1) Data selection: The trained base model $f(\theta, \mathbf{V})$ is used to predict a new instance $\mathbf{v}^{(i)}$, and based on the prediction score of the model, i.e if $f(\theta, \mathbf{v}^{(i)}) < \alpha$, the instance will be selected into a data batch B'. The offline DCG solution is computed for each $\mathbf{v}^{(i)} \in B'$ to obtain the scheduling structure $\mathbf{x}^{(i)}$, while keeping the application-level input $\hat{\mathbf{v}}^{(i)}$ to be used for model updating.

2) Model updating: The AID model retraining is preceded by the re-clustering of the scheduling structures $\mathbf{x}^{(i)} \in \mathcal{X}$, where $\mathcal{X} = \{\mathcal{X}_{old} \cup \mathcal{X}_{new}\}$ and $\mathcal{X}_{new} = \{\mathbf{x}^{(i)}, \cdots, \mathbf{x}^{(|B'|)}\}$. The Dunn's index algorithm is applied to find the optimal number of clusters to be used for re-clustering, this is because it is difficult to determine if $\mathcal{X}_{new}$ will be grouped to the existing K clusters, or if $\mathcal{X}_{new}$ will cause a reduction or increase in the number of clusters as shown in Algorithm 2.

The pre-trained weights of the base model $f(\theta, \mathbf{V})$ are used as starting points in the updating of $\hat{f}(\theta, \mathbf{V})$ in a transfer learning manner. Note that the last fully-connected layer of $\hat{f}(\theta, \mathbf{V})$ is different from that of $f(\theta, \mathbf{V})$ because of the updated number of classes from the clustering stage.

Since we do not want the AID model to always (or arbitrarily) update when a new instance $\mathbf{v}^{(i)} \in B'$ is observed, we define a model update trigger τ, with the condition $|B'| \geq \tau$, where τ is the minimum batch size[4] required. After every self-renewal process, the data batch B' becomes empty, i.e $|B'| = 0$. The self-renewal process can be lifelong or end when the model has learned on sufficient data.

Algorithm 1: Data Selection for Model Updating

Input: Base model $f(\theta, \mathbf{V})$, confidence score α, test set $\mathcal{B} = \{\mathbf{v}^{(i)}, \cdots, \mathbf{v}^{(|\mathcal{B}|)}\}$, where $\mathbf{v}^{(i)} = \{\mathbf{x}^{(i)}, \hat{\mathbf{v}}^{(i)}\}$
Output: $B', \mathcal{X}_{new}, \mathcal{V}_{new}$, where $B' \neq \emptyset$
for $\boldsymbol{v}^{(i)}$ ***in*** $\mathcal{B}$ **do**
  /* $\mathbf{x}^{(i)}$ is scheduling structure, $\hat{\mathbf{v}}^{(i)}$ is application-level input */
  if $\hat{\boldsymbol{v}}^{(i)} \rightarrow f(\theta, \hat{\boldsymbol{v}}^{(i)}) < \alpha$ **then**
    $B' \leftarrow \mathbf{v}^{(i)}; \mathcal{X}_{new} \leftarrow \mathbf{x}^{(i)}; \mathcal{V}_{new} \leftarrow \hat{\mathbf{v}}^{(i)}$
  end
end
return $B', \mathcal{X}_{new}, \mathcal{V}_{new}$

Algorithm 2: AID Model Retraining

Input: Clustering data $\mathcal{X}_{old}$, base model $f(\theta, \mathbf{V})$, old Dunn's index DN_{old}, data batch B', and old data $\mathcal{V}_{old}$
Output: $\hat{f}(\theta, \mathbf{V})$, where $|\mathcal{V}| \geq |\mathcal{V}_{old}|$
/* Re-clustering the data */
$\mathcal{X} = \{\mathcal{X}_{old} \cup \mathcal{X}_{new}\}$;
for *k=2* **to** *K* **do**
  Dunn's index algorithm on $\mathcal{X}$;
  if $DN_{new} > DN_{old}$ *or* $DN_{new} < DN_{old}$ **then**
    update K;
    do Kmeans++ clustering
  end
  else
    do K-nearest neighbor on $\mathcal{X}_{new}$
  end
end
/* Model updating */
$\mathcal{V} = \{\mathcal{V}_{old} \cup \mathcal{V}_{new}\}$;
load pre-trained weights $f(\theta, \mathbf{V})$;
update $\hat{f}(\theta, \mathbf{V}) \leftarrow f(\theta, \mathbf{V})$ using combined data $\mathcal{V}$;
return $\hat{f}(\theta, \boldsymbol{V})$

[4]This should not be misinterpreted as the batch size hyperparameter in Table II used for training a neural network.

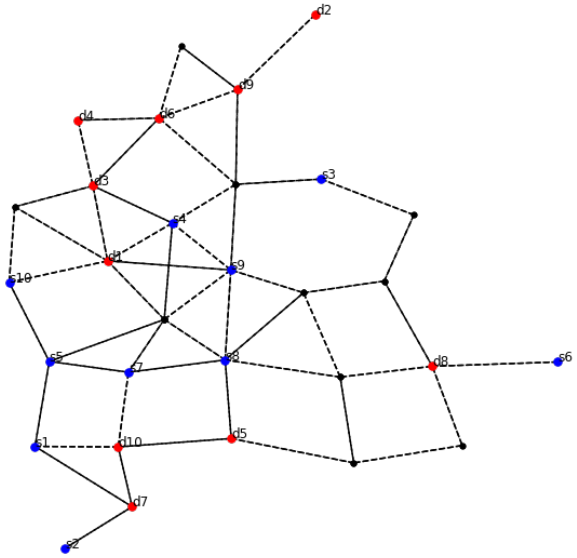

Fig. 3: A 30-node network topology with 10 commodity flows: source (blue) and destination (red).

IV. Numerical Experiments and Results

The implementation details and the results of the proposed SRML method are presented in this section.

A. Experiment Settings

We simulated a small-scale and large-scale multi-hop wireless network of 30 nodes and 100 nodes respectively, using a Python simulation environment, where the nodes are randomly distributed within a fixed 1 Km by 1 Km square area, and with a minimum separation distance of 30 m. The protocol interference model is used by uniformly setting the communication range and interference range of nodes as 50 m and 70 m respectively. Each link capacity $e_{m,n}^{cap}$ is calculated based on the Shannon formula using a channel bandwidth, preset signal power and noise power of 20 MHz, 17 dBm and -127 dBm respectively [2], with a path loss exponent of 3.5. The 30 nodes network topology is shown in Figure 3.

The experiments are run on Google Colaboratory (or Colab), equipped with the Nvidia K80 GPUs and Intel Xeon processors @2.3GHz. Numpy [20] and Tensorflow [21] packages are used in the Python environment.

B. Dataset Generation and Distribution

The dataset is generated using the DCG algorithm to obtain the optimization instances and their scheduling structures, which are used for the AID model training and clustering respectively. For the 30 nodes network and 100 nodes network, a total of 25,000 instances and 45,000 instances are computed as training data respectively, spanning 1 flow up to 5 flows and uniformly distributed. The testing set consists of 1000 instances, each, for 1 flow to 5 flows. However, in addition to the testing set, larger flows of 6 flows to 10 flows (each

TABLE III: Performance comparison between FML and proposed SRML method for 30 nodes network.

No. of Flows	1	2	3	4	5	6	7	8	9	10
t_{DCG}	1.97	2.49	2.61	2.90	2.98	3.10	8.0	9.51	10.11	10.68
CTR_{FML}	0.76	0.89	0.86	0.89	0.82	0.86	0.98	0.95	0.96	0.94
CTR_{SRML}	0.75	0.96	0.92	0.85	0.87	0.89	0.87	0.97	0.96	0.96
SOL_{DCG}	3.47	5.52	7.10	8.24	9.24	9.99	10.96	12.04	12.48	12.92
AR_{FML}	1.0	0.99	0.98	0.98	0.97	0.81	0.79	0.78	0.77	0.77
AR_{SRML}	1.0	0.99	0.98	0.98	0.97	0.87	0.83	0.82	0.80	0.79
SE_{FML}	154	243	308	358	399	359	382	419	428	444
SE_{SRML}	153	242	308	355	397	382	404	438	443	449

TABLE IV: Performance comparison between FML and proposed SRML method for 100 nodes network.

No. of Flows	1	2	3	4	5	6	7	8	9	10
t_{DCG}	18.55	33.0	53.55	44.12	67.16	54.06	83.06	64.93	93.29	79.69
CTR_{FML}	0.96	0.96	0.97	0.96	0.97	0.96	0.97	0.96	0.97	0.97
CTR_{SRML}	0.95	0.95	0.96	0.95	0.96	0.95	0.96	0.95	0.96	0.96
SOL_{DCG}	3.94	6.06	7.82	9.37	10.62	11.60	12.52	13.54	13.90	15.02
AR_{FML}	1.0	1.0	1.0	1.0	0.99	0.79	0.80	0.78	0.78	0.78
AR_{SRML}	1.0	1.0	1.0	1.0	0.99	0.86	0.86	0.83	0.82	0.82
SE_{FML}	97	150	194	232	260	227	245	262	267	288
SE_{SRML}	96	149	193	229	257	244	264	276	279	301

containing 1000 instances) are computed to evaluate the generalization capability of the SRML method against the FML method, RAND method and Greedy Heuristic method.

1) FML method: This is based on the isolated learning approach where a model is trained only on the available data and no further retraining or updating is performed when a new/future data is observed. The FML method is used in [2].

2) RAND method: This is a random self-renewal approach without the data selection strategy in Algorithm 1. With the RAND method, the data for model retraining is selected randomly as they are observed.

3) Greedy Heuristic method: This refers to the blind selection of a class of UCISs obtained from clustering, to solve an optimization instance.

For the initial clustering of the scheduling structures of the training data, $K = 1000$ for the small-scale and large-scale networks. During the inference phase using the trained base model, the test instances with prediction score below α (where $\alpha = 0.4$ in our experiment) are selected into batch B', which are combined with the original training data during the self-renewal of the ML model. We set $|B'| = 500$ and $|B'| = 1200$ for the 30 nodes network and 100 nodes network respectively (the update trigger τ is $\leq |B'|$). In the self-renewal stage, the number of scheduling classes are updated to $K = 1001$ and $K = 999$ for the 30 nodes network and 100 nodes network respectively, based on the Dunn's index algorithm.

C. Performance Metrics

We evaluated the performance of the proposed SRML framework using three metrics defined as follows:

1) Approximation ratio (AR): This is the ratio of the average achievable throughput (in Mbps) of the SRML method to that obtained by the DCG algorithm, computed as: $\frac{SOL_{\text{SRML}}}{SOL_{\text{DCG}}}$.

2) Computation time reduction (CTR): This is the ratio of the computation time difference between the DCG algorithm and SRML method to the computation time of the DCG algorithm, computed as: $\frac{t_{\text{DCG}} - t_{\text{SRML}}}{t_{\text{DCG}}}$. The t_{SRML} is the sum of the LP solution time and ML inference time, i.e $t_{\text{SRML}} = t_{\text{LP}} + t_{\text{INF}}$. The unit of time is in seconds.

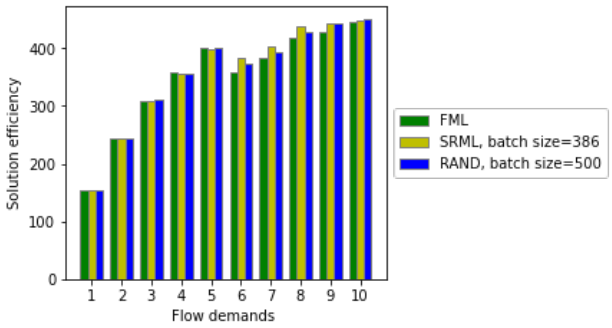

Fig. 4: Solution efficiency comparison in 30 nodes network.

3) Solution efficiency (SE): This is the ratio of the average achievable throughput of the SRML method to the total amount of training samples, measured in bits per second per sample, and computed as: $\frac{SOL_{\text{SRML}}}{|\mathcal{V}_{train}|}$, where $|\mathcal{V}_{train}|$ is the size of the training set.

The AR and CTR metrics are used in [2], but we introduce the SE in this paper to prove the supremacy of the SRML method over the FML method. Also, the SE metric reveals the benefit of our data selection algorithm for model retraining.

D. Results and Discussions

The performance of the SRML method is compared with the FML method as shown in Table III and Table IV for the 30 nodes network and 100 nodes network respectively. Similar to the SRML method, we allow the FML method to benefit from the Dunn's index algorithm in determining the optimal number of scheduling classes to group the scheduling structures of the training instances. However, despite such generosity, the SRML method outperforms the FML method in the 30 nodes network and 100 nodes network. For the AR, the SRML is better than the FML on all the flow demands, and very obvious

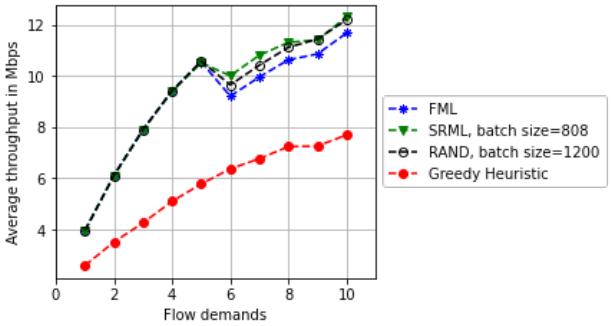

Fig. 5: Throughput comparison for 100 nodes network.

in the larger commodity flows from 6 flows to 10 flows which are the challenging cases. For 6 flows, this is 87% and 86%, and for 10 flows it is 79% and 82% for the 30 nodes network and 100 nodes network respectively. In contrast, for the FML, the AR for 6 flows is 81% and 79%, and for 10 flows it is 77% and 78% for the 30 nodes network and 100 nodes network respectively.

It should be noted that for the SRML method, we only performed one round of model updating and retraining with only very few instances, which we further explain as follows:

1) Effect of data selection on solution efficiency: With the data selection strategy, extracting the scheduling structures from only a small fraction of all the newly observed data can improve the solution efficiency. As shown in Figure 4, the SRML method achieves a higher SE than the FML method on larger flows by using only 386 instances for model updating, and this is better than the RAND method which used up to 500 instances. There is no compromise on the SRML performance on lower commodity flows, as it achieves a tightly close SE to the FML method. The last two rows in Table III and Table IV show the SE for the 30 nodes network and 100 nodes network respectively.

2) Performance comparison on achievable throughput: The average achievable throughput using the SRML method stems from the data selection algorithm, and this is related to the SE, i.e the higher the SE, the higher the throughput. In Figure 5 we show the benefit of the SRML method to the average throughput. Again, there is an obvious difference between the FML method and SRML method in the larger flow regime, with the SRML method outperforming the FML method consistently. The effect of the data selection strategy is evident when compared with the RAND method. While the SRML method only used 808 instances to improve throughput performance in one round, the RAND method used up to 1200 instances, but still performed sub-optimally to the SRML method. This implies that the RAND method will require several rounds with more training instances to reach a solution that is close to the SRML method. The noticeable gap between the SRML method and the Greeady Heuristic method shows the strength of ML in predicting the correct class of UCISs to solve an optimization instance. Without the supervised learning stage of AID model training, blindly selecting a class of UCISs from the clustering stage will not solve the optimization instance as shown by the Greedy Heuristic method in Figure 5.

The reduction in computation time which is a benefit of the FML method is also evident in the SRML method as shown by the CTR in Table III and Table IV for the 30 nodes network and 100 nodes network respectively. With the transfer learning technique used for the model updating, the learning parameters is significantly reduced by 71.09% and 87.04% for the 30 nodes network and 100 nodes network respectively. This offers benefit in storing the model, as well as increases training speed.

V. Related Work

The concept of self-renewal ML (SRML) in this paper is strongly related to lifelong machine learning (LML) or continual learning which has been proposed in the recent few decades [14], [15] but its application to wireless networking is still in its infancy and, thus, deserves attention. Applications such as chatbots, physical mobile robots, and intelligent assistants interact with humans and systems. To have an improved user experience with these applications, there is need for a continual learning approach to enhance their capability, and this is based on self-motivation or self-learning. Lifelong learning cuts across different areas of ML such as supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [16], [22], [23].

Seminal works in supervised LML have proposed techniques such as memory-based learning and neural networks to facilitate a classifier's capability in learning new tasks from the knowledge of previous tasks [14]. The LML for neural networks can be extended to cumulative learning[5] [24] where a new multi-class classifier is trained by combining the old data with the new data. Other LML algorithms have been proposed in [25], [26] to improve multi-task learning for intelligent applications. Lifelong information extraction and lifelong relaxation labeling methods have been proposed in [16], [22] which follows the unsupervised LML paradigm. For semi-supervised LML, the work in [27] proposed a never-ending language learning learner (NELL) which began to read and accumulate millions of entities from the web since 2010.

In the context of wireless networking, some few recent works have considered LML or continual learning approaches for network intrusion detection [28], resource optimization [29] and channel estimation [30]. Since wireless networks are dynamic, the goal of LML is to gracefully handle tasks that continue to evolve in the network which classical machine learning algorithms cannot handle, as learning is usually isolated. The work in [28] examined the suitability of two continual learning algorithms, i.e elastic weight consolidation and gradient episodic memory, for the anomaly intrusion detection in a wireless network. The problem of resource allocation in a dynamic wireless environment motivate the work in [29] to propose a continual learning approach where

[5]This is similar to the steps in the SRML model retraining in this paper. With cumulative learning it is possible to encounter a new class in the data.

the learning model incrementally adapts to changing episodes in the network - the effectiveness of the continual learning model was evaluated. A similar study in [28] applied LML to the channel estimation problem in 5G millimeter-wave MIMO systems. While these papers scarcely considered the network capacity optimization issue, in this paper, we have proposed the self-renewal ML approach to address the resource allocation problems in wireless multi-hop networks by learning more knowledge incrementally.

VI. Conclusion

We have presented a self-renewal ML method for assisting the optimization of complex commodity flow deployments when the available data is limited. The selection of useful samples during model inference to increase learning at the retraining phase of the AID model shows a performance improvement in the approximation ratio and solution efficiency for a typical small-scale network and large-scale network. Lifelong machine learning is a promising area in wireless network optimization, as it helps to alleviate the problem of limited knowledge due to insufficient available data, which is predominant in wireless networking. The self-renewal methodology in this paper can be extended to other lower layer or upper layer problems in the network protocol stack.

Acknowledgments

This work was supported in part by NSF grants CNS-1816908 and CNS-2008092.

References

[1] X. Lin and N. B. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, vol. 2, 2004, pp. 1484–1489.

[2] S. Zhang, O. T. Ajayi, and Y. Cheng, "A self-supervised learning approach for accelerating wireless network optimization," *IEEE Transactions on Vehicular Technology*, 2023.

[3] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu, "Impact of interference on multi-hop wireless network performance," in *Proceedings of the 9th annual international conference on Mobile computing and networking*, 2003, pp. 66–80.

[4] Y. Cheng, X. Cao, X. Shen, D. M. Shila, and H. Li, "A systematic study of the delayed column generation method for optimizing wireless networks," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, 2014, pp. 23–32.

[5] D. Chafekar, V. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Approximation algorithms for computing capacity of wireless networks with sinr constraints," in *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, 2008, pp. 1166–1174.

[6] S. Misra, S. D. Hong, G. Xue, and J. Tang, "Constrained relay node placement in wireless sensor networks: Formulation and approximations," *IEEE/ACM Transactions on networking*, vol. 18, no. 2, pp. 434–447, 2009.

[7] R. Gandhi, Y.-A. Kim, S. Lee, J. Ryu, and P.-J. Wan, "Approximation algorithms for data broadcast in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 11, no. 7, pp. 1237–1248, 2012.

[8] M. Pan, P. Li, and Y. Fang, "Cooperative communication aware link scheduling for cognitive vehicular networks," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 4, pp. 760–768, 2012.

[9] G. Chen, X. Cao, L. Liu, C. Sun, and Y. Cheng, "Joint scheduling and channel allocation for end-to-end delay minimization in industrial wirelesshart networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2829–2842, 2018.

[10] S. Zhang, B. Yin, and Y. Cheng, "Deep reinforcement learning for scheduling in multi-hop wireless networks," in *2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, 2021, pp. 1–9.

[11] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *ieee journal on selected areas in communications*, vol. 37, no. 6, pp. 1248–1261, 2019.

[12] L. Liu, B. Yin, S. Zhang, X. Cao, and Y. Cheng, "Deep learning meets wireless network optimization: Identify critical links," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 167–180, 2018.

[13] S. Zhang, B. Yin, W. Zhang, and Y. Cheng, "Topology aware deep learning for wireless network optimization," *IEEE Transactions on Wireless Communications*, vol. 21, no. 11, pp. 9791–9805, 2022.

[14] B. Liu, "Lifelong machine learning: a paradigm for continuous learning," *Frontiers of Computer Science*, vol. 11, pp. 359–361, 2017.

[15] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–207, 2018.

[16] Q. Liu, B. Liu, Y. Zhang, D. S. Kim, and Z. Gao, "Improving opinion aspect extraction using semantic similarity and aspect associations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[17] N. Ilc, "Modified dunn's cluster validity index based on graph theory," *Przeglad Elektrotechniczny*, vol. 88, no. 2, pp. 126–131, 2012.

[18] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on information theory*, vol. 46, no. 2, pp. 388–404, 2000.

[19] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[20] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[21] S. S. Girija, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *Software available from tensorflow. org*, vol. 39, no. 9, 2016.

[22] L. Shu, B. Liu, H. Xu, and A. Kim, "Separating entities and aspects in opinion targets using lifelong graph labeling," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2016.

[23] H. B. Ammar, E. Eaton, J. M. Luna, and P. Ruvolo, "Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[24] G. Fei, S. Wang, and B. Liu, "Learning cumulatively to become more knowledgeable," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1565–1574.

[25] A. Kumar and H. Daume III, "Learning task grouping and overlap in multi-task learning," *arXiv preprint arXiv:1206.6417*, 2012.

[26] P. Ruvolo and E. Eaton, "Ella: An efficient lifelong learning algorithm," in *International conference on machine learning*, 2013, pp. 507–515.

[27] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka, and T. Mitchell, "Toward an architecture for never-ending language learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 24, no. 1, 2010, pp. 1306–1313.

[28] S. K. Amalapuram, A. Tadwai, R. Vinta, S. S. Channappayya, and B. R. Tamma, "Continual learning for anomaly based network intrusion detection," in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, 2022, pp. 497–505.

[29] H. Sun, W. Pu, X. Fu, T.-H. Chang, and M. Hong, "Learning to continuously optimize wireless resource in a dynamic environment: A bilevel optimization perspective," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1900–1917, 2022.

[30] S. Kumar, S. K. Vankayala, B. S. Sahoo, and S. Yoon, "Continual learning-based channel estimation for 5g millimeter-wave systems," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 2021, pp. 1–6.