Learning-Assisted Algorithm Unrolling for Online Optimization with Budget Constraints

Jianyi Yang and Shaolei Ren
University of California, Riverside
{jyang239, shaolei}@ucr.edu

Abstract

Online optimization with multiple budget constraints is challenging since the online decisions over a short time horizon are coupled together by strict inventory constraints. The existing manually-designed algorithms cannot achieve satisfactory average performance for this setting because they often need a large number of time steps for convergence and/or may violate the inventory constraints. In this paper, we propose a new machine learning (ML) assisted unrolling approach, called `LAAU` (Learning-Assisted Algorithm Unrolling), which unrolls the agent's online decision pipeline and leverages an ML model for updating the Lagrangian multiplier online. For efficient training via backpropagation, we derive gradients of the decision pipeline over time. We also provide the average cost bounds for two cases when training data is available offline and collected online, respectively. Finally, we present numerical results to highlight that `LAAU` can outperform the existing baselines.

Introduction

Online optimization with budget (or inventory) constraints, also referred to as OOBC, is an important problem modeling a wide range of sequential decision-making applications with limited resources, such as online virtual machine resource allocation (Joe-Wong et al. 2013; Palomar and Chiang 2007), one-way trading in economics (El-Yaniv et al. 2001), resource management in wireless networks (Neely 2010; Lan et al. 2010), and data center server provisioning (Ghodsi et al. 2011). More specifically, when virtualizing a physical server into a small number of virtual machines (VMs) to satisfy the demand of multiple sequentially-arriving jobs, the agent must make sure that the total VM resource consumption is no more than what the physical server can provide (VMware 2022).

In an OOBC problem, online actions are selected sequentially to maximize the total utility over a short time horizon while the resource consumption over the time horizon is strictly constrained by a fixed amount of budgets (i.e., violating the budget constraint is naturally prohibited due to physical constraints). Consequently, the *short* time horizon (e.g., 24 hourly decisions in a day) and the *strict* budget constraint present substantial algorithmic challenges — the optimal solution relies on complete offline context information, but in the online setting, only the online contexts are revealed and the exact future contexts are unavailable for decision making (Lin et al. 2019, 2022).

A relevant but different problem is online optimization with (long-term) constraints (Balseiro, Lu, and Mirrokni 2020; Feldman et al. 2010; Neely 2010; Azar et al. 2016). In the literature, a common approach is to relax the long-term capacity constraints and include them as additional weighted costs into the original optimization objective, i.e., Lagrangian relaxation (Devanur et al. 2019; Zinkevich 2003; Balseiro, Lu, and Mirrokni 2020; Neely 2010). The Lagrangian multiplier can be interpreted as the resource *price* (Palomar and Chiang 2007), and is updated at each time step by a manually-designed algorithm such as Dual Mirror Descent (DMD) (Balseiro, Lu, and Mirrokni 2020; Wei, Yu, and Neely 2020; Jiang, Li, and Zhang 2020). These algorithms require a sufficiently long time horizon for convergence, which hence may not provide satisfactory performance for short-term budget constraints, especially when contexts in an episode are not identically independently distributed (i.i.d.). Additionally, some studies consider constraints on average (i.e., equivalently, long-term constraints) (Qiu et al. 2018; He et al. 2013) or bound the violation of the constraints (Neely 2010; Yu and Neely 2020; Sun et al. 2016). Thus, they do not apply to *strict* budget constraints over a short time horizon.

The challenges of OOBC with short-term and strict budget constraints can be further highlighted by that competitive online algorithms have only been proposed very recently under settings with linear constraints (Lin et al. 2019, 2022). Concretely, CR-Pursuit algorithms are proposed to make actions by following a pseudo-optimal algorithm based on the competitive ratio pursuit framework. Nonetheless, to make sure the solution exists for each OOBC episode, the guaranteed competitive ratio (ratio between the algorithm cost and the offline-optimal cost) can be large. Also, they treat each OOBC problem instance as a completely new one and focus on the worst-case competitive ratio without considering the available historical data obtained when solving previous OOBC episodes. Thus, their conservative nature does not result in a satisfactory average performance, which may limit the practicability of these algorithms.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

By tapping into the power of historical data, a natural idea for OOBC is to train an machine learning (ML) based optimizer. Indeed, reinforcement learning has been proposed to solve online allocation problems in other contexts (Kong et al. 2019; Alomrani, Moravej, and Khalil 2021; Du, Wu, and Huang 2019). But, the existing ML-based algorithms for online optimization typically learn online actions in an end-to-end manner without exploiting the structure of the online problem being studied, which hence can have an unnecessarily high learning complexity and create additional challenges for generalization to unseen problem instances (Chen et al. 2021; Liu et al. 2019).

Contribution. We study OOBC with short-term and strict budget constraints, and propose a novel ML-assisted unrolling approach based on recurrent architectures, called `LAAU` (Learning-Assisted Algorithm Unrolling). Instead of using an end-to-end ML model to directly learn online actions, `LAAU` uniquely exploits the `LAAU` problem structure and unrolls the agent's online decision pipeline into decision pipeline with three stages/layers — update the Lagrangian multiplier, optimize decisions subject to constraints, and update remaining resource budgets — and only plugs an ML model into the first stage (i.e., update the Lagrangian multiplier) where the key bottleneck for better performance exists. Thus, compared with the end-to-end model, `LAAU` benefits generalization by exploiting the knowledge of decision pipeline (Chen et al. 2021). Moreover, when the action dimension is larger than the number of constraints (i.e., the dimension of Lagrangian multipliers), the complexity advantage of using `LAAU` to learn Lagrangian multipliers can be further enhanced compared to learning the actions using an end-to-end model.

It is challenging to train `LAAU` through backpropagation since the constrained optimization layer is not easily differentiable. Thus, we derive tractable gradients for back-propagation through the optimization layer based on Karush-Kuhn-Tucker (KKT) conditions. In addition, we rigorously analyze the performance of `LAAU` in terms of the expected cost for both the case when the offline distribution information is available and the case when the data is collected online. Finally, to validate `LAAU`, we present numerical results by considering online resource allocation for maximizing the weighted fairness metric. Our results highlight that `LAAU` can significantly outperform the existing baselines and is very close to the optimal oracle in terms of the fairness utility.

Related Works

Constrained online optimization. Some earlier works (Devanur and Hayes 2009; Feldman et al. 2010) solve online optimization with (long-term) constraints by estimating a fixed Lagrangian multiplier using offline data. This approach works only for long-term or average constraints. Many other studies design online algorithms by updating the Lagrangian multiplier in an online style (Devanur et al. 2019; Balseiro, Lu, and Mirrokni 2020; Wei, Yu, and Neely 2020; Jiang, Li, and Zhang 2020). These algorithms guarantee sub-linear regrets under the i.i.d. context setting, and thus can achieve high utility if the number of time steps is sufficiently large. Likewise, the Lyapunov optimization approach addresses the long-term packing constraints by introducing virtual queues (equivalent to the Lagrangian multiplier) (Neely 2010; Yu and Neely 2020; Huang, Liu, and Hao 2014). Nonetheless, it also requires a sufficiently large number of time steps for convergence. By contrast, we consider online optimization with short-term strict budget constraints, which, motivated by practical applications, makes OOBC significantly more challenging.

Our work is relevant to the studies on OOBC (Lin et al. 2019, 2022) which design online algorithms to achieve a worst-case performance guarantee. However, to guarantee the worst-case performance and the feasibility of the algorithm, the algorithms are very conservative and their average performances are unsatisfactory. Comparably, we consider a more general setting where the budget constraint can be nonlinear, and utilize available historical data more efficiently to design ML-based `LAAU` that unrolls the online decision pipeline and achieves favorable average performance.

Algorithm unrolling. `LAAU` is related to the recent studies on ML-assisted algorithm unrolling and deep implicit layers, which integrate ML into traditional algorithmic frameworks for better generalization and interpretability, lower sampling complexity and/or smaller ML model size (Adler and Öktem 2018; Chen et al. 2021; Kolter, Duvenaud, and Johnson 2022; Monga, Li, and Eldar 2021; Liu et al. 2019). Algorithm unrolling has been used for sparse coding (Gregor and LeCun 2010; Sprechmann, Bronstein, and Sapiro 2015), signal and image processing (Monga, Li, and Eldar 2021; Li et al. 2019), and solving inverse problems (Kobler et al. 2020) and ordinary differential equations (ODEs) (Chen et al. 2018). Also, algorithm unrolling is applied in *learning to optimize* (L2O) (Chen et al. 2021; Wichrowska et al. 2017). Among these works, (Narasimhan et al. 2020) predicts the Lagrangian multiplier by a model to efficiently solve offline optimizations which may have a large number of constraints but allow constraint violations. These studies have their own challenges orthogonal to our problem where the key challenge is the lack of complete offline information. Thus, `LAAU`, to our knowledge, is the first to leverage ML to unroll an online optimizer for solving the online convex optimization with budget constraints, thus having better generalization than generic RL-based optimizers to directly obtain end solutions (Alomrani, Moravej, and Khalil 2021; Du, Wu, and Huang 2019; Kong et al. 2019).

Problem Formulation

As in the existing ML-based optimizers for online problems (Kong et al. 2019; Alomrani, Moravej, and Khalil 2021; Du, Wu, and Huang 2019), we consider an agent that interacts with a stochastic environment. The time horizon of an episode consists of N time steps. For an episode, two vectors $\boldsymbol{c} = [c_1, \cdots, c_N]^\top$ and $\boldsymbol{B} = [B_1, \cdots, B_M]^\top$, where c_t is a context vector and $B_m \in \mathbb{R}^+$ is the total budget for resource m, are drawn from a certain joint distribution $(\boldsymbol{c}, \boldsymbol{B}) \sim \mathcal{P}$, which we refer to as the environment distribution. Note that c_i and c_j for $i \neq j$ can follow different probability distributions, and so can B_i and B_j for $i \neq j$. The random vector $\boldsymbol{B} = [B_1, \cdots, B_M]^\top$ are revealed at the beginning of an episode, and represents the budgets for M types of resources. On the other hand, $\boldsymbol{c} = [c_1, \cdots, c_N]^\top$ are online contexts sequentially revealed over N different steps within an episode. That is, at step t, the agent only knows $c_1, \cdots, c_t$, but *not* the future parameters $c_{t+1}, \cdots, c_N$.

At each step $t = 1, \cdots, N$, the agent makes a decision $x_t \in \mathbb{R}^d$, consumes some budgets, and also receives a utility. Given the decision x_t and parameter c_t, the amount of the resource consumption is denoted as a non-negative function $g_m(x_t, c_t) \geq 0$, for $m = 1, \cdots, M$. To be consistent with the notation of loss function, we use a *cost* or *loss* $l(x_t, c_t)$ to denote the *negative* of the utility — the less $l(x_t, c_t)$, the better. As the cost function $l(\cdot, c_t)$ is parameterized by c_t, knowing c_t is also equivalent to knowing the cost function. We assume that the loss function l and the constraint functions $g_m, m = 1, \cdots, M$ are twice continuously differentiable, and either the loss function l or one of the constraint functions $g_m, m = 1, \cdots, M$ is strongly convex in terms of the decision x_t.

For each episode with $(\boldsymbol{c}, \boldsymbol{B}) \sim \mathcal{P}$, the goal of the agent is minimizing its total cost over the N steps subject to M resource capacity constraints, which we formulate as follows:

$$\begin{aligned} &\min_{\boldsymbol{x}=(x_1,\cdots,x_N)} \sum_{t=1}^{N} l(x_t, c_t), \\ &\text{s.t.} \ \sum_{t=1}^{N} g_m(x_t, c_t) \leq B_m, m = 1, \cdots, M. \end{aligned} \tag{1}$$

This is an online optimization problem with inventory constraints (referred to as OOBC) in the sense that the short-term strict inventory constraints are imposed for each episode of N time steps. An episode has its own M capacity constraints which should be strictly satisfied, and the unused budgets cannot roll over to the next episode. For ease of notation, given a policy π which maps available inputs to *feasible* actions, we denote $L(\pi) = \sum_{t=1}^{N} l(x_t, c_t)$ as the total loss for one episode and $\mathrm{E}[L(\pi)]$ as the expected total loss over the distribution of $(\boldsymbol{c}, \boldsymbol{B}) \sim \mathcal{P}$.

The setting of OOBC presents new technical challenges compared with existing works on constrained online optimization. Specifically in OOBC, the time horizon in an episode (episode length) is finite and can be very short. In this case, there are not many steps for algorithms to converge, and bad decisions at early steps have a large impact on the overall performance. Thus, DMD (Balseiro, Lu, and Mirrokni 2020; Wei, Yu, and Neely 2020; Jiang, Li, and Zhang 2020) and Lyapunov optimization (Neely 2010) which are specifically designed for long episodes may not provide good results for OOBC. Besides, unlike some studies that satisfy average constraints (Qiu et al. 2018; He et al. 2013) or that only approximately satisfy the constraints under bounded violations (i.e., *soft* constraints) (Yu and Neely 2020, 2019; Sun et al. 2016), OOBC requires all the constraints in Eqn. (1) be strictly satisfied. This requirement is necessary for many practical applications with finite available resources (e.g., a data center's power capacity must not be exceeded (Fan, Weber, and Barroso 2007)), but makes the problem more challenging. Last but not least, the contexts in one episode in OOBC are drawn from a general joint distribution (not necessarily i.i.d.). Under non-i.i.d. cases, DMD(Balseiro, Lu, and Mirrokni 2020) has performance guarantees only when each episode is long enough. CR-Pursuit(Lin et al. 2019, 2022) has competitive ratios but is too conservative and may not perform well on average. To improve the average performance of OOBC, new algorithms are needed to effectively utilize history data of previous episodes.

Learning-Assisted Algorithm Unrolling

Relaxed Optimization

The design of LAAU is based on the Lagrangian relaxed optimization method which is introduced here. Since it is difficult to directly solve the constrained optimization in Eqn. (1) due to the lack of complete offline information in an online setting, many studies (Arora, Hazan, and Kale 2012; Balseiro, Lu, and Mirrokni 2020; Neely 2010) solve the Lagrangian relaxed form written as follows:

$$\min_{\boldsymbol{x}=(x_1,\cdots,x_N)} \sum_{t=1}^{N} l(x_t, c_t) + \lambda^\top \sum_{t=1}^{N} \boldsymbol{g}(x_t, c_t), \tag{2}$$

where $\boldsymbol{g}(x_t, c_t) = [g_1(x_t, c_t), \cdots, g_M(x_t, c_t)]^\top$ and $\lambda = [\lambda_1, \cdots, \lambda_M]^\top$ is the non-negative Lagrangian multiplier corresponding to the M constraints $\sum_{t=1}^{N} \boldsymbol{g}(x_t, c_t) \leq \boldsymbol{B}$. The multiplier λ with M dimensions essentially relaxes the M inventory constraints, thus decoupling the decisions over the N time steps within an episode. It is also interpreted as the resource *price* in the resource allocation literature (Palomar and Chiang 2007; Boyd, Boyd, and Vandenberghe 2004; Neely 2010): A greater λ_t means a higher price for the resource consumption, thus pushing the agent to use less resource. Clearly, had we known the optimal Lagrangian multiplier λ^* at the beginning of each episode, the OOBC problem would become very easy. Unfortunately, knowing λ^* also requires the complete offline information $(\boldsymbol{c}, \boldsymbol{B})$, which is not possible in the online case. Nevertheless, if we can appropriately update λ_t in an online manner while strictly satisfying the constraints, we can also efficiently solve the OOBC problem. Formally, by using λ_t that is updated online for each step t, we can instead solve the following relaxed problem:

$$\min_{x_t} l(x_t, c_t) + \lambda_t^\top \boldsymbol{g}(x_t, c_t), \ \text{s.t.,} \ \boldsymbol{g}(x_t, c_t) \leq \boldsymbol{b}_t, \tag{3}$$

where $\lambda_t = [\lambda_{t,1}, \cdots, \lambda_{t,M}]^\top$, $\boldsymbol{g}(x_t, c_t) = [g_1(x_t, c_t), \cdots, g_M(x_t, c_t)]^\top$, and the remaining budget for step t is $\boldsymbol{b}_t = \boldsymbol{B} - \sum_{s=1}^{t-1} \boldsymbol{g}(x_s, c_s)$.

In fact, designing good update rules for λ_t for each step $t = 1, \cdots, N$ is commonly considered in the literature (Balseiro, Lu, and Mirrokni 2020; Agrawal and Devanur 2014). For example, (Balseiro, Lu, and Mirrokni 2020) update λ_t by DMD, in order to meet *long*-term constraints while achieving a low regret compared to the optimal oracle. Nonetheless, it requires a large number of time steps to converge to a good Lagrangian parameter. Likewise, the

Algorithm 1: Online Inference Procedure of LAAU

```
Input: ML model $f_\theta$.
1: for t=1 to N do
2:     Receive $c_t$, forward propagate $f_\theta$ and get Lagrangian multiplier $\lambda_t = f_\theta\left(\boldsymbol{b}_t, c_t, \bar{t}\right)$.
3:     Solve the constrained convex optimization in (4) and make action $x_t$.
4:     Update the resource budget $\boldsymbol{b}_{t+1} = \boldsymbol{b}_t - \boldsymbol{g}(x_t, c_t)$.
5: end for
```

Lyapunov optimization technique introduces a virtual queue, whose length essentially takes the role of λ_t and is updated as $\lambda_{t+1} = \max\left\{\lambda_t + \boldsymbol{g}(x_t, c_t) - \frac{1}{N}\boldsymbol{B}, 0\right\}$ or in other similar ways (Neely 2010). Nonetheless, the convergence rate of using Lyapunov optimization is slow (even assuming c_t is i.i.d. for $t = 1, \cdots, N$), and there exists a tradeoff between cost minimization and long-term constraint satisfactory, making it unsuitable for the short-term constraints that we focus on.

Alternatively, one may want to exploit the distribution information of $(\boldsymbol{c}, \boldsymbol{B}) \sim \mathcal{P}$ and solve a relaxed problem offline by considering M average constraints (referred to as AVG-LT). That is, we replace the short-term capacity constraints in Eqn. (1) with $\mathrm{E}_{\mathcal{P}}\left[\sum_{t=1}^{N} g_m(x_t, c_t)\right] \leq B_m$ for $m = 1, \cdots, M$. By solving this relaxed problem, we can obtain a Lagrangian multiplier $\lambda_{\mathcal{P}}$ that only depends on $\mathcal{P}$ but not the specific $(\boldsymbol{c}, \boldsymbol{B})$. Thus, we can replace λ_t in Eqn. (3) with $\lambda_{\mathcal{P}}$. However, since this method uses a constant Lagrangian multiplier for all episodes, we will either be overly conservative and not using the budgets as much as possible, or violating the the constraints.

Algorithm Unrolling

We propose to leverage the powerful capacity of ML to find a solution. One approach is to train an end-to-end model that takes the online input information and directly outputs a decision. But, the end-to-end model should be large enough to capture the possibly complex logic of the optimal policy, and the end-to-end models often have poor interpretability and worse generalization (see the comparison between LAAU and the generic end-to-end approach in Section). Therefore, instead of replacing the whole decision pipeline with ML, *we only plug an ML model in the most challenging stage* — online updating of the Lagrangian multiplier λ_t needed to solve the relaxed problem in Eqn. (3).

As shown in Algorithm 1 and illustrated in Fig. 1, the decision pipeline at step t can be decomposed into three stages as follows.

Updating λ_t. At the beginning of step $t = 1, \cdots, N$, the ML model takes the parameter c_t, the remaining budget $\boldsymbol{b}_t = [b_{t,1}, \cdots, b_{t,m}]^\top$ and the normalized number of remaining steps $\bar{t} = \frac{N-t}{N}$ as the inputs, and outputs the Lagrangian multiplier $\lambda_t = [\lambda_{t,1}, \cdots, \lambda_{t,m}]^\top$. Letting f_θ denote the ML model parameterized by θ, we have $\lambda_t = f_\theta\left(\boldsymbol{b}_t, c_t, \bar{t}\right)$.

Optimization layer. In the optimization layer, we solve a relaxed convex problem formulated in Eqn. (3). The natural

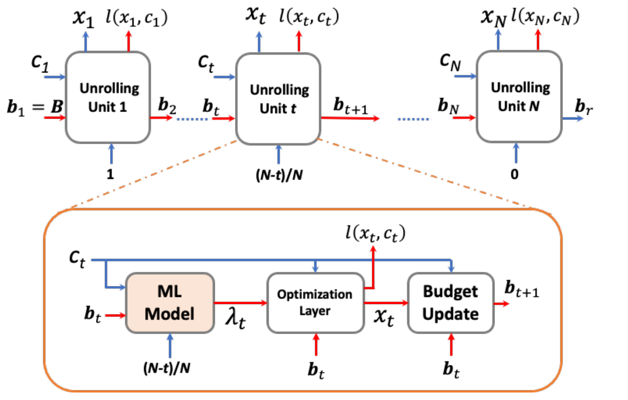

Figure 1: Architecture of LAAU. The red lines indicate the flows that need back propagation.

constraints on the remaining resource budgets ensure that the *strict* inventory constraints are always satisfied by LAAU. We denote the optimization layer as $p(c_t, \lambda_t, \boldsymbol{b}_t)$ and thus have:

$$x_t = p(c_t, \lambda_t, \boldsymbol{b}_t) = \underset{x}{\mathrm{argmin}}\left(l(x, c_t) + \lambda_t^\top \boldsymbol{g}(x, c_t)\right), \quad \text{s.t., } \boldsymbol{g}(x, c_t) \leq \boldsymbol{b}_t. \tag{4}$$

Updating resource budgets. In the last stage, the remaining resource budgets serve as an input for the next recurrence and are updated as $\boldsymbol{b}_{t+1} = \boldsymbol{B} - \sum_{s=1}^{t} \boldsymbol{g}(x_s, c_s) = \boldsymbol{b}_t - \boldsymbol{g}(x_t, c_t)$..

Each episode includes N recurrences, each for one decision step. The cost for step t is calculated after the optimization layer as $l(x_t, c_t)$. Note that in the N-th recurrence which is the final stopping step, the remaining budget $\boldsymbol{b}_r = \boldsymbol{b}_N - \boldsymbol{g}(x_N, c_N)$ will be wasted if not used up. Thus, we directly set $\lambda_N = 0$ for the N-th unrolling unit.

Training the Unrolling Architecture

To clearly explain the back propagation of the unrolling model, we first consider the offline training where offline distribution information is available. Then we extend to the online training setting where the data is collected online.

Offline Training

Training Objective For the ease of notation, we denote the online optimizer as $\boldsymbol{h}_\theta\left(\boldsymbol{B}, \boldsymbol{c}\right)$. For offline training, we are given an *unlabeled* training dataset $S = \{(\boldsymbol{c}_1, \boldsymbol{B}_1), \cdots, (\boldsymbol{c}_n, \boldsymbol{B}_n)\}$, with n samples of $(\boldsymbol{c}, \boldsymbol{B})$. The training dataset can be synthetically generated by sampling from the target distribution for the online input $(\boldsymbol{c}, \boldsymbol{B})$, which is a standard technique in the context of *learning to optimize* (Chen et al. 2021; Dai et al. 2017; Li and Malik 2017; Alomrani, Moravej, and Khalil 2021; Du, Wu, and Huang 2019). By forward propagation, we can get the empirical training loss as $L(\boldsymbol{h}_\theta, S) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{N} l(\boldsymbol{x}_{i,t}, \boldsymbol{c}_{i,t})$ where $\boldsymbol{x}_{i,t}$ is the output of the online optimizer $\boldsymbol{h}_\theta$ regarding $\boldsymbol{c}_i$ and $\boldsymbol{B}_i$. By minimizing the empirical loss, we get $\hat{\theta} = \arg\min_\theta L(\boldsymbol{h}_\theta, S)$.

Backpropagation Typically, the minimization of the training loss is performed by gradient descent-based algorithms like SGD or Adam, which need back propagation to get the gradient of the loss with respect to the ML model weight θ. Nonetheless, unlike standard ML training (e.g., neural network training with only linear and activation operations), our unrolled recurrent architecture includes an *implicit* layer — the optimization layer (Kolter, Duvenaud, and Johnson 2022). Additionally, the unrolling architecture has multiple skip connections. Thus, the back-propagation process is dramatically different from that of standard recurrent neural networks. Next, we derive the gradients for back propagation in our unrolling design. Note that the loss $l(x_t, c_t)$ for any $t = 1, \cdots, N$ is directly determined by the output of the optimization layer x_t and the parameter c_t, and x_t needs back propagation. Thus, by the chain rule, we have

$$\triangledown_\theta l(x_t, c_t) = \triangledown_{x_t} l(x_t, c_t)(\triangledown_{\lambda_t} x_t \triangledown_\theta \lambda_t + \triangledown_{\boldsymbol{b}_t} x_t \triangledown_\theta \boldsymbol{b}_t). \quad (5)$$

To get $\triangledown_{\lambda_t} x_t$ and $\triangledown_{\boldsymbol{b}_t} x_t$ in Eqn. (5), we need to perform back propagation for the optimization layer $p(c_t, \lambda_t, \boldsymbol{b}_t)$. This is a challenging task and will be addressed in Section . The other gradients in Eqn. (5) include $\triangledown_\theta \lambda_t$ and $\triangledown_\theta \boldsymbol{b}_t$. Note that λ_t, which is the ML model output directly determined by its ML model weight θ, and the remaining budget $\boldsymbol{b}_t$ both need back propagation. Thus, the gradient of λ_t with respect to the ML model weight θ is expressed as

$$\triangledown_\theta \lambda_t = \triangledown_\theta f_\theta(\boldsymbol{b}_t, c_t, \bar{t}) + \triangledown_{\boldsymbol{b}_t} f_\theta(\boldsymbol{b}_t, c_t, \bar{t}) \triangledown_\theta \boldsymbol{b}_t, \quad (6)$$

Now, it remains to derive $\triangledown_\theta \boldsymbol{b}_t$, which is important since b_t is the signal connecting two adjacent recurrences. By the expression of b_t in Line 4 of Algorithm 1, we have

$$\triangledown_\theta \boldsymbol{b}_t = \triangledown_\theta \boldsymbol{b}_{t-1} + \triangledown_{x_{t-1}} \boldsymbol{g}(x_{t-1}, c_{t-1}) \triangledown_{\lambda_{t-1}} x_{t-1} \triangledown_\theta \lambda_{t-1}, \quad (7)$$

Combining Eqn. (5), (6) and (7), we get the recurrent expression for back propagation. Then, by adding up the gradients of the losses over N time steps, we get the gradient of the total loss as $\triangledown_\theta L(\boldsymbol{h}_\theta, S) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{N} \triangledown_\theta l(x_{i,t}, c_{i,t})$.

Differentiating the Optimization Layer It is challenging to get the close-form solution and its gradients for many constrained optimization problems. One possible remedy is to use some black-box gradient estimators like zero-order optimization (Liu et al. 2020; Ruffio et al. 2011). However, zero-order gradient estimators are not computationally efficient since many samples are needed to estimate a gradient. Another method is to train a deep neural network to approximate the optimization layer in Eqn. (4) and then calculate the gradients based on the neural network. However, we need many samples to pre-train the neural network, and the gradient estimation error can be large. To address these challenges, we analytically differentiate the solution to Eqn. (4) in the optimization layer with respect to the inputs λ_t, and $\boldsymbol{b}_t$ by exploiting KKT conditions (Kolter, Duvenaud, and Johnson 2022; Boyd, Boyd, and Vandenberghe 2004). The KKT-based differentiation method, given in Proposition 0.1, is computationally efficient, explainable and accurate (under mild technical conditions).

Proposition 0.1 (Back-propagation by KKT). *Assume that x_t and μ_t are the primal and dual solutions to Eqn.* (4) *, respectively. Let* $\Delta_{11} = \triangledown_{x_t x_t} l(x_t, c_t) + \sum_{m=1}^{M}(\lambda_{m,t} + \mu_{m,t}) \triangledown_{x_t x_t} g_m(x_t, c_t)$, $\Delta_{12} = [\triangledown_{x_t} \boldsymbol{g}(x_t, c_t)]^\top$, $\Delta_{21} = \mathrm{diag}(\mu_t) \triangledown_{x_t} \boldsymbol{g}(x_t, c_t)$, *and* $\Delta_{22} = \mathrm{diag}(\boldsymbol{g}(x_t, c_t) - \boldsymbol{B}_t)$. *If the conditions in Proposition 0.2 are satisfied, the gradients of the optimization layer w.r.t.* λ_t *and* $\boldsymbol{b}_t$ *are*

$$\triangledown_{\lambda_t} x_t = -\left(\Delta_{11}^{-1} + \Delta_{11}^{-1}\Delta_{12}\mathrm{Sc}(\Delta, \Delta_{11})^{-1}\Delta_{21}\Delta_{11}^{-1}\right)\Delta_{12},$$

$$\triangledown_{\boldsymbol{b}_t} x_t = -\Delta_{11}^{-1}\Delta_{12}\mathrm{Sc}(\Delta, \Delta_{11})^{-1}\mathrm{diag}(\mu_t),$$

where $\mathrm{Sc}(\Delta, \Delta_{11}) = \Delta_{22} - \Delta_{21}\Delta_{11}^{-1}\Delta_{12}$ *denotes the Shur-complement of* Δ_{11} *in* $\Delta = [[\Delta_{11}, \Delta_{12}]; [\Delta_{21}, \Delta_{22}]]$. □

We find that to get truly accurate gradient computation by Proposition 0.2, the Shur-complement $\mathrm{Sc}(\Delta, \Delta_{11})$ and Δ_{11} should be invertible. Otherwise, we can get approximated gradients by taking pseudo-inverse of $\mathrm{Sc}(\Delta, \Delta_{11})$ and Δ_{11}. The sufficient conditions to guarantee perfectly accurate gradient computation are given in Proposition 0.2.

Proposition 0.2 (Sufficient Conditions of Accurate Differentiation)**.** *Assume that the problem in Eqn.* (4) *satisfies strong duality. The loss l or one of the constraints $g_m, m = 1, \cdots, M$ is strongly convex with respect to x. Denote $\mathcal{A}$ as the index set of constraints that are activated (i.e., equality holds) under the optimal solution. $\mu_{m,t} \neq 0, \forall m \in \mathcal{A}$, the size of the activation set satisfies $|\mathcal{A}| \leq d$ with d as the action dimension, and the gradients $\triangledown_{x_t} g_m(x_t), m \in \mathcal{A}$ are linearly independent and not zero vectors, the gradients in Proposition 0.1 are perfectly accurate.*

Remark 1. To derive the gradient of Eqn. (4) with respect to an input parameter, we take gradients on both sides of the equations in KKT conditions by the chain rule and get new equations about the gradients. By solving the obtained set of equations and exploiting the block matrix inversion, we can derive the gradients with respect to the inputs in Proposition 0.1.

The conditions in Proposition 0.2 are mild in practice. First, strong duality is easily satisfied for the considered convex optimization in Eqn. (4) given the Slater's condition (Boyd, Boyd, and Vandenberghe 2004). Besides, the requirement of strong convexity excludes linear programming (LP). Actually, LP problems with resource constraints are usually solved by other relaxations other than our considered relaxation in Eqn. (2) (Devanur et al. 2019). The other conditions are related to activated constraints. According to the condition of complementary slackness (Boyd, Boyd, and Vandenberghe 2004), the condition that optimal dual variables corresponding to the activated constraints are not zero typically holds. We also require that the number of activated constraints is less than the action dimension, and the gradient vectors of the activated constraint functions under optimal solutions should be independent from each other. Given that at most a small number of constraints are activated in most cases, the two conditions are easily satisfied. Actually, the independence condition requires that the activated constraints are not redundant — an activated constraint function is not a linear combination of any other activated constraint functions; otherwise, it can be replaced by other constraints.

Online Training

In practice, we may have a cold-start setting without many offline samples. An efficient approach for this setting is online stochastic gradient descent (SGD) with its algorithm in Appendix. Concretely, when the i-th instance arrives, we perform online inference by Algorithm 1. After the instance with N steps ends, we collect the context and budget data of this episode and update the ML model weight $\hat{\theta}_i$ by performing one-step gradient descent, i,e, $\hat{\theta}_i = \hat{\theta}_{i-1} - \bar{\alpha}\nabla_\theta L(h_{\hat{\theta}_{i-1}}, \boldsymbol{c}_i)$ where $L(h_{\hat{\theta}_{i-1}}, \boldsymbol{c}_i)$ is the loss of the unrolling model for the ith instance and $\bar{\alpha}$ is the stepsize. The back propagation method is the same as the offline training. Then, with the updated $\hat{\theta}_i$, we perform inference by Algorithm 1 for the instance in the $(i+1)$-th round. We will show by analysis that the average cost decreases with time.

Performance Analysis

In this section, we bound the expected cost when the trained ML model f_θ is used in LAAU.

Definition 1. The weight in the ML model f_θ (and also the online optimizer $\boldsymbol{h}_\theta$) that minimizes the expected loss $\mathrm{E}\left[L(\boldsymbol{h}_\theta, \boldsymbol{c})\right]$ with respect to the distribution of $(\boldsymbol{c}, \boldsymbol{B}) \sim \mathcal{P}$ is defined as $\theta^* = \arg\min_{\theta\in\Theta} \mathrm{E}\left[L(\boldsymbol{h}_\theta, \boldsymbol{c})\right]$, and the weight that minimizes the empirical loss $L(\boldsymbol{h}_\theta, S)$ is defined as $\hat{\theta}^* = \arg\min_{\theta\in\Theta} L(\boldsymbol{h}_\theta, S)$, where Θ is the weight space.

In Definition 1, given the weight space Θ, $\boldsymbol{h}_{\theta^*}$ is the best online optimizer based on the unrolling architecture in terms of the expected cost. $\boldsymbol{h}_{\theta^*}$ is not the offline-optimal policy, but it is close to the policy that performs best given available online information when the capacity of the ML model and weight space Θ are large enough. Next, we show the performance gap of LAAU compared with h_{θ^*}.

Theorem 0.3. *By the optimization layer in Eqn.* (4), *LAAU satisfies the inventory constraints for each OOBC instance. Suppose that $\hat{\theta}$ is the ML model weight by offline training on dataset S with n samples, and that we plug it into the online optimizer $\boldsymbol{h}_{\hat{\theta}}$. With probability at least $1-\delta, \delta \in (0,1)$,*

$$\begin{aligned}\mathrm{E}\left[L(\boldsymbol{h}_{\hat{\theta}})\right] - \mathrm{E}\left[L(\boldsymbol{h}_{\theta^*})\right] \le \mathcal{E}\left(\boldsymbol{h}_{\hat{\theta}}, S\right) + 4R_n(L\circ\mathbb{H}) \\ +2\left(\Gamma_{L,c}\omega_c + \Gamma_{L,b}\omega_b\right)\sqrt{\frac{\ln(2/\delta)}{n}},\end{aligned} \tag{8}$$

where $\mathcal{E}\left(\boldsymbol{h}_{\hat{\theta}}, S\right) = L(\boldsymbol{h}_{\hat{\theta}}, S) - L(\boldsymbol{h}_{\hat{\theta}^*}, S)$ is the training error, $R_n(L\circ\mathbb{H})$ is the Rademacher complexity regarding the loss space $L\circ\mathbb{H} = \{L(\boldsymbol{h}), \boldsymbol{h}\in\mathbb{H}\}$ with $\mathbb{H}$ being the ML model set, $\omega_c = \max_{\boldsymbol{c},\boldsymbol{c}'\in\mathbb{C}}\|\boldsymbol{c}-\boldsymbol{c}'\|$ is the size of the parameter space $\mathbb{C}$, $\omega_b = \max_{\boldsymbol{B},\boldsymbol{B}'\in\mathbb{B}}\|\boldsymbol{B}-\boldsymbol{B}'\|$ is the size of the capacity constraint space $\mathbb{B}$, $\Gamma_{L,c}$ and $\Gamma_{L,b}$ are the Lipschitz constants of the total loss $L(\boldsymbol{h}_\theta, \boldsymbol{c}) = \sum_{t=1}^{N} l(x_t, c_t)$ with respect to $\boldsymbol{c}$ and $\boldsymbol{B}$, respectively.

Proposition 0.4. *If a linear model $f_\theta(\boldsymbol{v}) = \theta^\top\phi(\boldsymbol{v}), \|\theta\| \le Z$ is used as the ML model in LAAU, the Rademacher complexity $R_n(L\circ\mathbb{H})$ is bounded ny $O\left(\frac{ZW}{\sqrt{n}}\right)$, where $W = \sup_{\boldsymbol{v}}\sqrt{\phi(\boldsymbol{v})^\top\phi(\boldsymbol{v})}$. If a neural network, where the depth is K, the width is less than u, activation functions are Γ_α-Lipschitz continuous, and the spectrum norm of the weight matrix in layer k with is less than Z_k, is used as the ML model, the Rademacher complexity $R_n(L\circ\mathbb{H})$ is bounded by $R_n(L\circ\mathbb{H}) \le O\left(\frac{K^{3/2}u\Gamma_\alpha(\beta_b+\beta_c)\prod_{k=1}^{K} Z_k}{\sqrt{n}}\right)$, where β_b, β_c are the largest l_2-norm of $\boldsymbol{B}$ and $\boldsymbol{c}$. The notation O in this proposition indicates the scaling relying on M, N, the Lipschitz constants of loss function l, constraint function g, optimization layer p and neural network f.*

Remark 2. Theorem 0.3 shows that the performance gap between LAAU and the pseudo-oracle h_{θ^*} in terms of expected loss is bounded by the empirical training error, plus a generalization error which relies on the Radmacher complexity, the LipSchitz constant of the online optimizer, and the number of training samples. The Radmacher complexity indicates the richness of the loss function space with respect to the online optimizer space $\mathbb{H}$ and the distribution $\mathcal{P}$ and is further bounded in Proposition 0.4. From the bound of Radmacher complexity, we find that for both linear model and neural network, the generalization error increases with episode length N and the number of constraints M. Besides, the Rademacher complexity relies on the ML model designs. For example, if a linear model is used as the ML model, the generalization error relies on the norm bounds of feature mapping and linear weights, while if a neural network is used as the ML model, the generalization error is related to the network length , width, the smoothness of activation functions, and spectral norm bounds of the weights in each layer. The last term of the expected cost bound is scaled by $(\Gamma_{L,c}\omega_c + \Gamma_{L,b}\omega_b)$ which indicates the sensitivity of the loss when the inputs are changed. This term highly depends on the Lipschitz constants of the total loss regarding the two inputs. Our proof in the appendix gives the bound of the Lipschitz constants of the loss regarding its inputs. □

Proposition 0.5 (Average Cost of Online Training). *Assume that for each round i, $\nabla_\theta\mathrm{E}[L(h_{\hat{\theta}_i})]$ is $\Gamma_{\nabla L,\theta}$-Lipschitz continuous, and the Polyak-Lojasiewicz inequality is satisfied, i.e. $\exists\varsigma > 0, \left\|\nabla_\theta\mathrm{E}[L(h_{\hat{\theta}_i})]\right\|^2 \ge 2\varsigma\left(\mathrm{E}[L(h_{\hat{\theta}_i})] - \mathrm{E}[L(h_{\theta^*})]\right)$. Also, assume that for the distribution of $\boldsymbol{c}_i$,, $\exists\iota_G > \iota > 0$ such that $\forall i$, $\left\langle\nabla_\theta\mathrm{E}[L(h_{\hat{\theta}_i})], \mathrm{E}\left[\nabla_\theta L(h_{\hat{\theta}_i}, \boldsymbol{c}_i)\right]\right\rangle \ge \iota\left\|\nabla_\theta\mathrm{E}[L(h_{\hat{\theta}_i})]\right\|_2^2$, $\|\mathrm{E}[\nabla_\theta L(h_{\hat{\theta}_i}, \boldsymbol{c}_i)]\|_2 \le \iota_G\left\|\nabla_\theta\mathrm{E}[L(h_{\hat{\theta}_i})]\right\|_2$, and there exist $\varpi, \varpi_V > 0$ such that $\forall i$, $\mathrm{Var}\left[\nabla_\theta L(h_{\hat{\theta}_i}, \boldsymbol{c}_i)\right] \le \varpi + \varpi_V\left\|\nabla_\theta\mathrm{E}[L(h_{\hat{\theta}_i})]\right\|_2^2$. Then with the same notations as Theorem 0.3, for the online setting where LAAU is trained by SGD with stepsize $0 < \bar{\alpha} \le \frac{\iota}{\Gamma_{\nabla L,\theta}(\varpi_V + \iota_G^2)}$, with probability at least $1-\delta, \delta\in(0,1)$, we have for each round i*

$$\mathrm{E}\left[L(\boldsymbol{h}_{\hat{\theta}_i}) - L(\boldsymbol{h}_{\theta^*})\right] \le \frac{\bar{\alpha}\Gamma_{\nabla L,\theta}\varpi}{2\iota\varsigma} + O\left((1-\bar{\alpha}\iota\varsigma)^i\right), \tag{9}$$

where the expectation is taken over the randomness of context $\boldsymbol{c}$ and budget $\boldsymbol{B}$ and the model weight $\hat{\theta}_i$ by SGD.

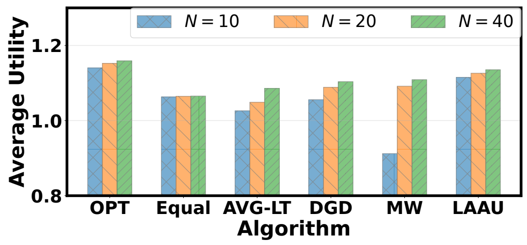

Figure 2: Average utility with different episode lengths

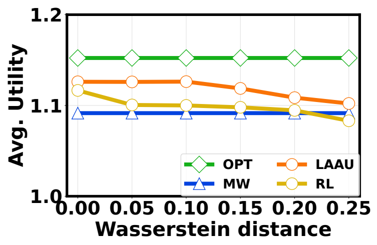

Figure 3: Avg. utility with different Wasserstein distances.

Proposition 0.5 bounds the expected loss gap between the learned weight $\hat{\theta}_i$ by SGD and the optimal weight θ^* in Definition 1. The first non-reducible term is caused by the randomness of the context and budget $\{(\boldsymbol{c}_i, \boldsymbol{B}_i)\}$. The second term decreases with time, and the convergence rate depends on the sequence randomness and the parameter ς in the Polyak-Lojasiewicz inequality.

Numerical Results

Weighted fairness is a classic performance metric in the resource allocation literature (Lan et al. 2010), including fair allocation in computer systems (Ghodsi et al. 2011) economics (Hylland and Zeckhauser 1979). Here, we consider a general *online* setting. A total of N jobs arrive sequentially, and job t has a weight $c_t \geq 0$. The agent allocates resource $x_t \geq 0$ to job t at each step t. We consider the commonly-used weighted fairness $\sum_{t=1}^{N} c_t \log(x_t)$ (Lan et al. 2010). We create the training and testing samples based on the Azure cloud workload dataset, which contains the average CPU reading for tasks at each step (Shahrad et al. 2020).

We consider several baseline algorithms as follows. The Offline Optimal Oracle **OPT** is the solution to the problem in Eqn. (1). We consider two heuristics: One is Equal Resource Allocation (**Equal**) which equally allocates the total resource capacity to N jobs, and another one is Resource Allocation with Average Long-term Constraints (**AVG-LT**) which relaxes the inventory constraints of the weighted fairness problem as $\mathrm{E}_{\mathcal{P}}\left[\sum_{t=1}^{N} x_t\right] \leq B$ and uses the optimal Lagrangian multiplier for this relaxed problem as λ_t for online allocation. We consider two algorithms based on dual mirror descent which are Dual Gradient Descent (**DGD**) and Multiplicative Weight (**MW**) (Balseiro, Lu, and Mirrokni 2020). To reduce the resource waste after the last step, we slightly revise DGD and MW by setting the allocation decision for job N as $\min(b_N, x_{\max})$. CR-pursuit (**CR-pursuit**) is the state-of-the-art online algorithm that makes online actions by tracking a pseudo-optimal algorithm with a competitive guarantee (Lin et al. 2019, 2022). We also compare LAAU with the end-to-end Reinforcement Learning (**RL**). A neural network with the same size of the ML model as in LAAU is used in RL to directly predict the solution x_t, given parameter c_t and budget b_t as inputs.

Average utility. We first show in Fig. 2 the average utilities (per time step). We do not add the average utility of CR-pursuit in the figure because its average utility for our evaluation instances is as low as 0.562, exceeding the utility range of the figure. Clearly, OPT achieves the highest utility, but it is infeasible in practice due to the lack of complete offline information. We can observe that the average utilities by expert algorithms including AVG-LT, DGD, MW are even below the average utility by the simple equal allocation (Equal) when the episode length is $N = 10$. This is because all the three algorithms are designed for online optimizations with long-term constraints, and not suitable for the more challenging short-term counterparts. By contrast, LAAU performs well for all cases with large and small episode lengths, and outperforms the other algorithms designed for long-term constraints even when N is as large as 40. This demonstrates the power of LAAU in solving challenging online problems with inventory constraints. More results are given in the appendix.

OOD testing. In practice, the training-testing distributional discrepancy is common. We measure the training-testing distributional difference by the Wasserstein distance d_W. We choose the setting with episode length $N = 20$ to perform the OOD evaluation. To create the distributional discrepancy, we add i.i.d. Gaussian noise with different means and variances to the training data and keep the testing data the same as the default setting. The offline optimal and MW do not make use of the training distribution, and hence are not affected. We can see in Fig. 3 that, the OOD testing decreases the performance of both LAAU and RL, but LAAU is less affected by OOD testing than RL and is still higher than that of the baseline MW even under large Wasserstein distance. This is because the unrolling architecture in LAAU has an optimization layer and a budget update layer, which are deterministic and have no training parameters, so only the ML model to learn the Lagrangian multiplier is affected by OOD testing. Comparably, RL uses an parameterized end-to-end policy model trained on the offline data, so it has worse performance under OOD testing. This highlights that by the unrolling archietecture, LAAU has better generalization performance than end-to-end models.

Conclusion

In this paper, we focus on OOBC and propose a novel ML-assisted unrolling approach based on the online decision pipeline, called LAAU. It leverages an ML model for updating the Lagrangian multiplier online. We derive the gradients for back-propagation and bound the expected cost. Finally, we present numerical results on weighted fairness and highlight LAAU significantly outperforms the existing baselines in terms of the average performance.

Acknowledgements

This work was supported in part by the NSF under grant CNS-1910208.

References

Adler, J.; and Öktem, O. 2018. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 37(6): 1322–1332.

Agrawal, S.; and Devanur, N. R. 2014. Fast algorithms for online stochastic convex programming. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, 1405–1424.

Alomrani, M. A.; Moravej, R.; and Khalil, E. B. 2021. Deep Policies for Online Bipartite Matching: A Reinforcement Learning Approach. *CoRR*, abs/2109.10380.

Arora, S.; Hazan, E.; and Kale, S. 2012. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1): 121–164.

Azar, Y.; Buchbinder, N.; Chan, T. H.; Chen, S.; Cohen, I. R.; Gupta, A.; Huang, Z.; Kang, N.; Nagarajan, V.; Naor, J.; et al. 2016. Online algorithms for covering and packing problems with convex objectives. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, 148–157. IEEE.

Balseiro, S.; Lu, H.; and Mirrokni, V. 2020. Dual mirror descent for online allocation problems. In *International Conference on Machine Learning*, 613–628.

Boyd, S.; Boyd, S. P.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Chen, R. T.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. 2018. Neural ordinary differential equations. *NeurIPS*.

Chen, T.; Chen, X.; Chen, W.; Heaton, H.; Liu, J.; Wang, Z.; and Yin, W. 2021. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*.

Dai, H.; Khalil, E. B.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. In *NIPS*.

Devanur, N. R.; and Hayes, T. P. 2009. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings of the 10th ACM conference on Electronic commerce*, 71–78.

Devanur, N. R.; Jain, K.; Sivan, B.; and Wilkens, C. A. 2019. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. *Journal of the ACM (JACM)*, 66(1): 1–41.

Du, B.; Wu, C.; and Huang, Z. 2019. Learning resource allocation and pricing for cloud profit maximization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 7570–7577.

El-Yaniv, R.; Fiat, A.; Karp, R. M.; and Turpin, G. 2001. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1): 101–139.

Fan, X.; Weber, W.-D.; and Barroso, L. A. 2007. Power provisioning for a warehouse-sized computer. In *ISCA*.

Feldman, J.; Henzinger, M.; Korula, N.; Mirrokni, V. S.; and Stein, C. 2010. Online stochastic packing applied to display ad allocation. In *European Symposium on Algorithms*, 182–194.

Ghodsi, A.; Zaharia, M.; Hindman, B.; Konwinski, A.; Shenker, S.; and Stoica, I. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Nsdi*, volume 11, 24–24.

Gregor, K.; and LeCun, Y. 2010. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, 399–406.

He, J.; Xue, Z.; Wu, D.; Wu, D. O.; and Wen, Y. 2013. CBM: Online strategies on cost-aware buffer management for mobile video streaming. *IEEE Transactions on Multimedia*, 16(1): 242–252.

Huang, L.; Liu, X.; and Hao, X. 2014. The Power of Online Learning in Stochastic Network Optimization. *SIGMETRICS Perform. Eval. Rev.*, 42(1): 153–165.

Hylland, A.; and Zeckhauser, R. 1979. The efficient allocation of individuals to positions. *Journal of Political economy*, 87(2): 293–314.

Jiang, J.; Li, X.; and Zhang, J. 2020. Online Stochastic Optimization with Wasserstein Based Non-stationarity. *arXiv preprint arXiv:2012.06961*.

Joe-Wong, C.; Sen, S.; Lan, T.; and Chiang, M. 2013. Multi-resource Allocation: Fairness-efficiency Tradeoffs in a Unifying Framework. *IEEE/ACM Trans. Netw.*, 21(6): 1785–1798.

Kobler, E.; Effland, A.; Kunisch, K.; and Pock, T. 2020. Total deep variation: A stable regularizer for inverse problems. *arXiv preprint arXiv:2006.08789*.

Kolter, Z.; Duvenaud, D.; and Johnson, M. 2022. Deep Implicit Layers. http://implicit-layers-tutorial.org/. Accessed: 2022-01-01.

Kong, W.; Liaw, C.; Mehta, A.; and Sivakumar, D. 2019. A New Dog Learns Old Tricks: RL Finds Classic Optimization Algorithms. In *ICLR*.

Lan, T.; Kao, D.; Chiang, M.; and Sabharwal, A. 2010. An axiomatic theory of fairness in network resource allocation. In *INFOCOM*.

Li, K.; and Malik, J. 2017. Learning to Optimize. In *ICLR*.

Li, Y.; Tofighi, M.; Monga, V.; and Eldar, Y. C. 2019. An algorithm unrolling approach to deep image deblurring. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7675–7679. IEEE.

Lin, Q.; Mo, Y.; Su, J.; and Chen, M. 2022. Competitive Online Optimization with Multiple Inventories: A Divide-and-Conquer Approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(2): 1–28.

Lin, Q.; Yi, H.; Pang, J.; Chen, M.; Wierman, A.; Honig, M.; and Xiao, Y. 2019. Competitive online optimization under inventory constraints. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(1): 1–28.

Liu, R.; Cheng, S.; Ma, L.; Fan, X.; and Luo, Z. 2019. Deep proximal unrolling: Algorithmic framework, convergence analysis and applications. *IEEE Transactions on Image Processing*, 28(10): 5013–5026.

Liu, S.; Chen, P.-Y.; Kailkhura, B.; Zhang, G.; Hero III, A. O.; and Varshney, P. K. 2020. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5): 43–54.

Monga, V.; Li, Y.; and Eldar, Y. C. 2021. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2): 18–44.

Narasimhan, H.; Cotter, A.; Zhou, Y.; Wang, S.; and Guo, W. 2020. Approximate heavily-constrained learning with lagrange multiplier models. *Advances in Neural Information Processing Systems*, 33: 8693–8703.

Neely, M. J. 2010. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool.

Palomar, D. P.; and Chiang, M. 2007. Alternative Distributed Algorithms for Network Utility Maximization: Framework and Applications. *IEEE Trans. Automatic Control*, 52(12): 2254–2269.

Qiu, C.; Hu, Y.; Chen, Y.; and Zeng, B. 2018. Lyapunov optimization for energy harvesting wireless sensor communications. *IEEE Internet of Things Journal*, 5(3): 1947–1956.

Ruffio, E.; Saury, D.; Petit, D.; and Girault, M. 2011. Tutorial 2: Zero-order optimization algorithms. *Eurotherm School METTI.*

Shahrad, M.; Fonseca, R.; Goiri, Í.; Chaudhry, G.; Batum, P.; Cooke, J.; Laureano, E.; Tresness, C.; Russinovich, M.; and Bianchini, R. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 205–218.

Sprechmann, P.; Bronstein, A. M.; and Sapiro, G. 2015. Learning efficient sparse and low rank models. *IEEE transactions on pattern analysis and machine intelligence*, 37(9): 1821–1833.

Sun, Q.; Ren, S.; Wu, C.; and Li, Z. 2016. An online incentive mechanism for emergency demand response in geo-distributed colocation data centers. In *Proceedings of the seventh international conference on future energy systems*, 1–13.

VMware. 2022. Distributed Power Management Concepts and Use. http://www.vmware.com/files/pdf/Distributed-Power-Management-vSphere.pdf. Accessed:2022-01-01.

Wei, X.; Yu, H.; and Neely, M. J. 2020. Online primal-dual mirror descent under stochastic constraints. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2): 1–36.

Wichrowska, O.; Maheswaranathan, N.; Hoffman, M. W.; Colmenarejo, S. G.; Denil, M.; Freitas, N.; and Sohl-Dickstein, J. 2017. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, 3751–3760.

Yu, H.; and Neely, M. J. 2019. Learning-aided optimization for energy-harvesting devices with outdated state information. *IEEE/ACM Transactions on Networking*, 27(4): 1501–1514.

Yu, H.; and Neely, M. J. 2020. A Low Complexity Algorithm with $\mathcal{O}(\sqrt{T})$ Regret and $\mathcal{O}(1)$ Constraint Violations for Online Convex Optimization with Long Term Constraints. *Journal of Machine Learning Research*, 21(1): 1–24.

Zinkevich, M. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, 928–936.