A Lazy Approach to Online Learning with Constraints

Branislav Kveton
Intel Research
Santa Clara, CA
branislav.kveton@intel.com

Jia Yuan Yu
Department of Electrical and
Computer Engineering
McGill University
jia.yu@mcgill.ca

Georgios Theocharous
Intel Research
Santa Clara, CA
georgios.theocharous@intel.com

Shie Mannor
Department of Electrical and
Computer Engineering
McGill University
shie@ece.mcgill.ca

Abstract

In this paper, we study a sequential decision making problem. The objective is to maximize the total reward while satisfying constraints, which are defined at every time step. The novelty of the setup is our assumption that the rewards and constraints are controlled by a potentially adverse opponent. To solve the problem, we propose a novel expert algorithm that guarantees a vanishing regret while violating only some bounded number of constraints. The quality of our expert solutions is evaluated on a challenging power management problem. Results of our experiments show that online learning with constraints can be carried out successfully in practice.

Introduction

Online learning with expert advice (Cesa-Bianchi & Lugosi 2006) has been studied extensively by the machine learning community. The framework has been also successfully used to solve many real-world problems, such as adaptive caching (Gramacy *et al.* 2003) or power management (Helmbold *et al.* 2000; Dhiman & Simunic 2006; Kveton *et al.* 2007). The major advantage of the online setting is that no assumption is made about the environment. As a result, there is no need to build its model and estimate its parameters. In turn, this type of learning is naturally robust to environmental changes and suitable for solving dynamic real-world problems.

In this paper, we study online learning problems with side constraints. A similar setup was considered by Mannor and Tsitsiklis (2006). Side constraints are common in real-world domains. For instance, power management problems are often formulated as maximizing power savings subject to some average performance criteria. The criteria usually restrict the rate of bad power management actions and can be naturally represented by constraints.

Our work makes two contributions. First, we show how to apply prediction with expert advice to solve online optimization problems with constraints. Our solution is both practical and sound. Based on our knowledge, this is the first solution with such properties. Second, we use the proposed approach to solve a real-world power management (PM) problem.

The paper is structured as follows. First, we formulate our optimization problem and relate it to the existing work. Second, we propose and analyze a practical solution to the problem based on prediction with expert advice. Third, we evaluate the quality of our solution on a real-world PM problem. Finally, we summarize our work and suggest future research directions.

Online constrained optimization

In this paper, we study an online learning problem, where an agent wants to maximize its total reward subject to average-cost constraints. At every time t, the agent takes some action θ_t from the action set $\mathcal{A}$, and then receives a reward $r_t(\theta_t) \in [0, 1]$ and a cost $c_t(\theta_t) \in [0, 1]$. We assume that our agent has no prior knowledge on reward and cost functions except that they are bounded. Therefore, they can be generated in a non-stationary or even adverse way. The agent may consider only the past rewards $r_1, \ldots, r_{t-1}$ and the past costs $c_1, \ldots, c_{t-1}$ when deciding what action θ_t to take.

To clarify our online learning problem and its challenges, we first define an *offline* version of the problem. This offline version simply assumes that our agents knows all reward and cost terms in advance. In such a setting, the optimal strategy of the agent can be expressed as a solution to the constrained optimization problem:

$$\begin{aligned} \text{maximize}_{\boldsymbol{\theta}} \quad & \frac{1}{T} \sum_{t=1}^{T} r_t(\theta_t) \\ \text{subject to:} \quad & g_t(\boldsymbol{\theta}) \leq c_0 \quad \forall\, t \in \{1, \ldots, T\}\,; \end{aligned} \tag{1}$$

where the sequence of actions $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_T)$ is optimized to maximize the average reward over T time steps subject to T constraints. The constraints enforce the average cost $g_t(\boldsymbol{\theta})$ to be less than c_0 at every time step t, where $g_t(\boldsymbol{\theta})$ is defined

as a τ-step average of instantaneous costs:

$$g_t(\boldsymbol{\theta}) = \begin{cases} \frac{1}{t}\sum_{\ell=1}^{t} c_\ell(\theta_\ell) & \text{if } t < \tau \\ \frac{1}{\tau}\sum_{\ell=t-\tau+1}^{t} c_\ell(\theta_\ell) & \text{otherwise.} \end{cases} \quad (2)$$

This type of problems is common in the domains of financial mathematics, caching, or power management. For example, power management problems are often formulated as maximizing power savings subject to some performance criteria, which are averages over finite periods of time.[1]

The offline version of our online optimization problem (1) can be solved by standard techniques for nonlinear programming (Bertsekas 1999). In this work, we attempt to solve the problem online. The challenges of our online setting are that the horizon T is unknown, and no assumption on the rewards r_t and costs c_t is made in advance. As a result, it may be too ambitious to expect as good policies as in the offline setting. Therefore, our objective is more modest. We want to learn a policy that returns as high rewards as the best online solution to our problem chosen offline from some set of experts. This learning paradigm is known as prediction with expert advice (Littlestone & Warmuth 1994). At the same time, our online solution should satisfy all but a small number of constraints.

Existing work

This section positions our work with respect to the literature on online convex programming (Zinkevich 2003) and online learning with constraints (Mannor & Tsitsiklis 2006).

Online convex programming (Zinkevich 2003) involves a convex feasible set $\mathcal{F} \subset \mathbb{R}^n$ and a sequence of convex functions $f_t : \mathcal{F} \to \mathbb{R}$. At every time t, a decision maker chooses some action $\theta_t \in \mathcal{F}$ based on the past functions $f_1, \ldots, f_{t-1}$ and actions $\theta_1, \ldots, \theta_{t-1}$. The goal is to minimize the regret:

$$\sum_{t=1}^{T} f_t(\theta_t) - \min_{\theta \in \mathcal{F}} \sum_{t=1}^{T} f_t(\theta). \quad (3)$$

In contrast to online convex programming, the feasible set $\mathcal{F}$ in our problem (1) may change arbitrarily over time because our constraint functions (Equation 2) depend on the costs c_t. A useful interpretation is that the feasible set $\mathcal{F}$ is controlled by an adverse opponent. This is the main source of difficulty in online constrained optimization.

Mannor and Tsitsiklis (2006) investigated online learning in the context of the constrained optimization problem:

$$\begin{aligned} \text{maximize}_{\theta \in \Delta(\mathcal{A})} \quad & \frac{1}{T}\sum_{t=1}^{T} r_t(\theta) \quad (4) \\ \text{subject to:} \quad & \frac{1}{T}\sum_{t=1}^{T} c_t(\theta) \le c_0; \end{aligned}$$

where $\Delta(\mathcal{A})$ denotes the simplex of probability distributions over the set of actions $\mathcal{A}$. Based on their work, the reward in hindsight, which corresponds to the solution of the optimization problem (4), is generally unattainable online. Results in our paper do not contradict to this claim. The main reason is that the quality of our online policies is compared to weaker baselines. The baselines are expert policies that violate only a limited number of constraints. One of our major contributions is that we show how to build these policies in practice. Furthermore, note that the problem (4) involves only a single terminal constraint while our agent satisfies a new constraint at every time step (1).

Mannor and Tsitsiklis (2006) also suggested solving constrained optimization problems online based on a calibrated forecaster of the opponent's action. Unfortunately, the complexity of existing calibrated forecasting schemes makes this solution unattractive.

Online learning with constraints

In this section, we show how to learn an online policy $\boldsymbol{\theta}$ from a set of experts $\xi_1, \ldots, \xi_N$ that guarantees a sublinear regret:

$$\max_{n=1,\ldots,N} \sum_{t=1}^{T} r_t(\xi_n(t)) - \sum_{t=1}^{T} \mathrm{E}\left[r_t(\theta_t)\right] \quad (5)$$

with respect to the best of the experts, and a sublinear bound on the total number of constraint violations:

$$\sum_{t=1}^{T} \mathbf{1}_{[g_t(\boldsymbol{\theta}) > c_0]}. \quad (6)$$

In other words, we want to achieve close-to-optimal rewards as $T \to \infty$ and satisfy all but a limited number of constraints. Our approach is based on prediction with expert advice (Littlestone & Warmuth 1994), which is a standard online learning paradigm. To obtain guarantees on the regret, we employ a randomized regret minimizing algorithm over a pool of experts. To prove a sublinear bound on the constraint violation, we restrict the experts such that they violate only a sublinear number of constraints. Unfortunately, this is not sufficient to prove the bound because switching between the experts may generate additional constraint violations. Hence, we modify our regret minimizing algorithm to account for this problem. Before we discuss details of our solution, we explain how to build experts that violate only a small fraction of constraints.

Arbitration

Expert policies that violate only a small subset of constraints are the fundamental building blocks of our online solutions. This section illustrates how to build these policies efficiently by modifying ordinary policies. The rest of the paper simply assumes that such policies exist.

Our solution assumes that we have a special action θ^0 that guarantees $c_t(\theta^0) = 0$ for all time steps t. The key is to play the action θ^0 when the original policy may violate more than an acceptable fraction of constraints. We refer to this process as *arbitration*. An example of a simple arbitrator is provided in Figure 1. The arbitrator overrides actions of the policy $\boldsymbol{\theta}$ such that all average-cost constraints $g_t(\widehat{\boldsymbol{\theta}}) \le c_0$ are satisfied.

[1] This paper generalizes to constraint functions $g_t(\boldsymbol{\theta})$, which are arbitrary convex combinations of instantaneous costs c_t. Moreover, our optimization problem may involve multiple constraints at every time step t.

Inputs:
an arbitrated policy $\widehat{\boldsymbol{\theta}}$ up to the time $t-1$
an action θ_t suggested at the time step t
past cost functions $c_{t-\tau+1}, \ldots, c_{t-1}$

Algorithm:
if $\left(\frac{1}{\tau}\left(\sum_{\ell=t-\tau+1}^{t-1} c_\ell(\widehat{\theta}_\ell)+1\right) > c_0\right)$
choose an action $\widehat{\theta}_t = \theta^0$
else
choose an action $\widehat{\theta}_t = \theta_t$

Outputs:
an arbitrated policy $\widehat{\boldsymbol{\theta}}$ up to the time t

Figure 1: An arbitrator that overrides actions of the policy $\boldsymbol{\theta}$ such that all average-cost constraints $g_t(\widehat{\boldsymbol{\theta}}) \leq c_0$ are satisfied. Since the value of the variable $c_t(\theta_t)$ is unknown at the time of arbitration, we use its worst-case estimate $c_t(\theta_t) = 1$.

Arbitration can be carried out in most real-world domains. For instance, it corresponds to taking no power management actions in the power management domain. It is conceptually equivalent to having a non-empty feasible set in a traditional optimization setting. Unfortunately, the arbitrating action θ^0 often yields zero rewards. Therefore, it should be taken only when necessary because it conflicts with our main objective of maximizing rewards.

Finally, note that the greedy arbitrator (Figure 1) is a simple but rather conservative way of arbitrating policies. In the rest of this section, we discuss how to mix arbitrated policies online to obtain bounds on the regret and constraint violation of our solutions.

An illustrative example

First, let us consider an online learning setup where we have access to a pool of experts $\xi_1, \ldots, \xi_N$ that never violate constraints. Moreover, suppose that we apply the standard exponentially weighted forecaster (Cesa-Bianchi & Lugosi 2006) to the experts without taking constraints into account. Based on known results, the regret of the forecaster is bounded as:

$$\max_{n=1,\ldots,N} \sum_{t=1}^{T} r_t(\xi_n(t)) - \sum_{t=1}^{T} \mathrm{E}\left[r_t(\theta_t)\right] \leq \frac{\log(N)}{\eta} + \frac{\eta T}{2}, \quad (7)$$

where η is its learning rate, and the expectation is taken with respect to our randomized forecasting scheme. For $\eta = T^{-\frac{1}{2}}$, the bound is on the order of $O(\sqrt{T})$, and therefore sublinear in T.

Unfortunately, although the presented solution yields sublinear regret, it may not satisfy all constraints. The following example demonstrates this problem.

Inputs:
arbitrated expert policies $\xi_1, \ldots, \xi_N$
expert weights $w_{t-1}(1), \ldots, w_{t-1}(N)$
a reward function r_t
a learning rate η
an expert e_{t-1} followed at the time step $t-1$

Algorithm:
if $(t \pmod L) \equiv 1)$
randomly choose an expert e_t according to the distribution:
$P(e_t) = \frac{w_{t-1}(e_t)}{\sum_{n=1}^{N} w_{t-1}(n)}$
else
$e_t = e_{t-1}$

play an action $\theta_t = \xi_{e_t}(t)$

for every $n = 1, \ldots, N$
$w_t(n) = w_{t-1}(n) \exp[\eta r_t(\xi_n(t))]$

Outputs:
an action θ_t played at the time step t
updated expert weights $w_t(1), \ldots, w_t(N)$
an expert e_t followed at the time step t

Figure 2: An exponentially weighted forecaster that permits one expert switch per L time steps. We refer to the approach as lazy learning.

Example 1. *Suppose that ξ_1 and ξ_2 are expert policies that generate cost sequences:*

$$(c_t(\xi_1(t)))_{t=1,\ldots,T} = \frac{1}{5}(1,1,1,1,1,1,1,1,1,1,1,\ldots)$$

$$(c_t(\xi_2(t)))_{t=1,\ldots,T} = (0,0,0,0,1,0,0,0,0,1,0,\ldots).$$

Moreover, let our average-cost constraints be defined by the averaging window $\tau = 5$ and cost limit $c_0 = 0.2$. The values are set such that the experts ξ_1 and ξ_2 satisfy all constraints. Unfortunately, switching between the experts easily leads to constraint violations. For instance, if we follow the expert ξ_1 at the time $t = 1$ and switch to the expert ξ_2 at the next time step.

The above example illustrates that constraint violations may simply result from switching between expert policies. Since our average-cost constraints depend only on the most recent τ time steps, their violations may happen only within τ steps of the most recent expert switch. Therefore, the total number of the switching-induced violations can be bounded as:

$$\sum_{t=1}^{T} \mathbf{1}_{[g_t(\boldsymbol{\theta}) > c_0]} \leq U\tau, \quad (8)$$

where U is the number of expert switches. In the worst case, the bound is not sublinear in T since the number of switches may be proportional to T.

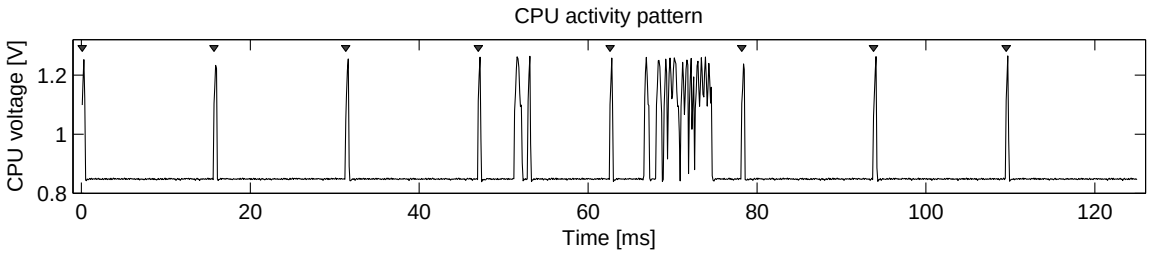

Figure 3: An example of a CPU activity pattern. The voltage is presented as a function of time (in milliseconds). Black triangles denote OS interrupts. Note that the distance between two consecutive OS interrupts is 15.6 ms. Due to this periodicity, software interrupts in Microsoft Windows can be easily predicted.

Lazy learning

To allow a sublinear bound on switching-induced violations, we alter the standard exponentially weighted forecaster such that it switches between expert policies infrequently relative to the time horizon. In particular, we partition the time steps $1, \ldots, T$ into T/L segments of the length L:

$$\begin{array}{ccc} (1, & \ldots, & L) \\ (L+1, & \ldots, & 2L) \\ \vdots & \vdots & \vdots \\ (T-L+1, & \ldots, & T) \end{array} \tag{9}$$

and permit expert switches at the beginning of each segment only. The parameter L can be chosen such that the maximum number of violations $(T/L)\tau$ (Equation 8) is sublinear in T, which was one of our objectives. It remains to show that the altered forecaster also guarantees a sublinear regret similarly to Equation 7.

The forecaster is described in Figure 2. We refer to it as a lazy learner because the pool of experts is greedily arbitrated to guarantee that no expert violates more than some bounded number of constraints.

Assumption 1. *Every expert policy ξ_n is allowed to violate at most x constraints. The assumption can be formalized as:*

$$\max_{n=1,\ldots,N} \sum_{t=1}^{T} \mathbf{1}_{[g_t(\xi_n)>c_0]} \leq x, \tag{10}$$

where the scalar x is potentially a function of T.

Under this assumption, we can prove the following bound on the performance of the forecaster.

Proposition 1. *Let $\xi_1, \ldots, \xi_N$ be expert policies that satisfy Assumption 1. Then the regret of the lazy learner (Figure 2) is bounded as:*

$$\max_{n=1,\ldots,N} \sum_{t=1}^{T} r_t(\xi_n(t)) - \sum_{t=1}^{T} \mathrm{E}\left[r_t(\theta_t)\right] \leq \frac{\log(N)}{\eta} + \frac{\eta T L}{2}.$$

Moreover, the number of constraints violated by the learner is bounded as:

$$\sum_{t=1}^{T} \mathbf{1}_{[g_t(\boldsymbol{\theta})>0]} \leq Nx + U\tau,$$

where U is the number of switches between the experts.

Proof: Our first claim is proved by interpreting lazy learning as a standard online learning problem:

$$\begin{aligned} &\max_{n=1,\ldots,N} \sum_{t=1}^{T} r_t(\xi_n(t)) - \sum_{t=1}^{T} \mathrm{E}\left[r_t(\theta_t)\right] \\ &= \max_{n=1,\ldots,N} \sum_{m=0}^{T/L-1} \sum_{t=mL+1}^{(m+1)L} \left(r_t(\xi_n(t)) - \mathrm{E}\left[r_t(\theta_t)\right]\right) \\ &\leq \frac{\log(N)}{\eta} + \frac{\eta}{2} \sum_{m=0}^{T/L-1} \\ &\quad \max_{n=1,\ldots,N} \left(\sum_{t=mL+1}^{(m+1)L} \left(r_t(\xi_n(t)) - \mathrm{E}\left[r_t(\theta_t)\right]\right) \right)^2 \\ &\leq \frac{\log(N)}{\eta} + \frac{\eta}{2} \frac{T}{L} L^2 \\ &= \frac{\log(N)}{\eta} + \frac{\eta T L}{2}. \end{aligned}$$

The first step of the proof follows by algebra, the second step follows from Theorem 1 (Cesa-Bianchi & Lugosi 2006), and the third step results from the reward terms r_t being bounded on the interval $[0, 1]$.

Our second claim follows from the construction of the algorithm. In particular, the term $U\tau$ provides an upper bound on the total number of constraint violations due to switching between the experts $\xi_1, \ldots, \xi_N$. Without taking the switching into account, the experts can together violate at most Nx constraints. Hence, the lazy learner cannot violate more than $Nx + U\tau$ constraints in total. ■

The above proposition demonstrates that we can bound both the regret and constraint violation of online learned policies. The parameters η and L can be chosen such that our bounds are sublinear in T. For instance, for $\eta = T^{-\frac{3}{4}}$ and $L = T^{\frac{1}{2}}$, the regret bound is on the order of $O(T^{\frac{3}{4}})$. To show that the constraint violation bound is sublinear in T at the same time, we note that $U \leq T/L$ and assume $x = T^{\frac{1}{2}}$. As a result, the number of constraint violations can be bounded on the order of $O(T^{\frac{1}{2}})$. Finally, note that the segment length L naturally allows for trading off the tightness of the two bounds.

In the rest of the paper, we evaluate the performance of the lazy learner on a power management problem. The nature of

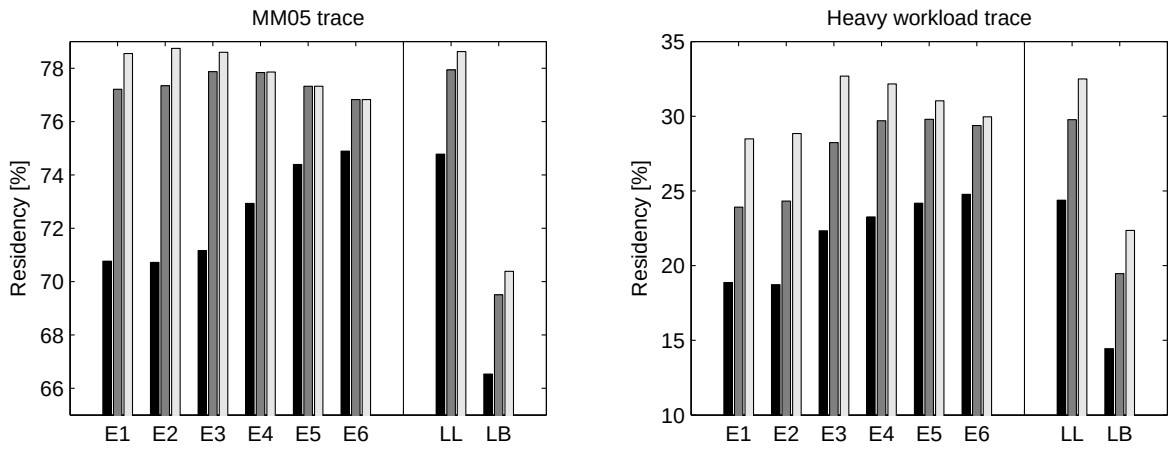

Figure 4: Comparison of lazily-learned policies (LL) to their pools of experts (E1, . . . , E6). The policies are compared by their average residency for three different latency budgets c_0: 0.02 (black bars), 0.04 (dark gray bars), and 0.06 (light gray bars). We also report lower bounds on the residency of the learned policies (LB) as suggested by Proposition 1. The evaluation is done on two CPU activity traces: MM05 and a heavy workload trace.

the problem is not adversarial as typically assumed in online learning. Therefore, although our approach learns almost as good policies as the best experts, our bounds are a little loose to justify its performance. This issue is discussed in detail in the experimental section.

Package power management

Our online solution is evaluated on a challenging real-world problem. We look at the power management of the complete processing unit including multi-core processors, L1 and L2 caches, and associated circuitry. Solving this PM problem is important because the complete processing unit may account for as much as 40 percent of the power consumed by mobile computers. In the rest of this paper, we use the term *package* to refer to the complete processing unit.

The primary goal of package PM is to minimize the power consumption of the package without impacting its perceived performance. This performance objective can be restated as maximizing the *residency* of the package in low power states while minimizing the *latency* in serving hardware interrupts. The latency is a delay caused by waking up the package from low power states. The central component of package PM is a prediction module, which predicts idle CPU periods that are sufficiently long to power down the package. This prediction is done at every OS interrupt. Under normal circumstances, Microsoft Windows generates OS interrupts periodically every 15.6 ms (Figure 3).

A state-of-the-art solution to package PM are static timeout policies. A *static timeout policy* (Karlin *et al.* 1994) is a simple power management strategy, which is parameterized by the timeout parameter T. When the package remains idle for more than T ms, the policy puts it into a low power state. When an unpredicted hardware interrupt occurs, the package must wake up to serve it. Due to the delay in performing this task, the package incurs a 1 ms latency penalty. The package wakes up ahead of the OS interrupts because these interrupts are predictable. This setting is suggested by domain experts.

In the experimental section, we consider a pool of experts, which are adaptive timeout policies. The policies adapt their timeout parameters at every OS interrupt (Figure 3) based on the current workload. Kveton *et al.* (2007) showed that they perform significantly better than static timeout policies when applied to package PM.

Experiments

The main goal of the experimental section is to demonstrate online learning with constraints in practice. Our experiments are performed on the package PM problem. We simulate the package in MATLAB on two CPU activity traces, which are collected from the Intel Core Duo CPU.

Experimental setup

The first trace is recorded during running MobileMark 2005 (MM05). MM05 is a performance benchmark that simulates the activity of an average Microsoft Windows user. A corresponding CPU activity trace is 90 minutes long and contains more than 500,000 OS interrupts. The second trace is generated by running Adobe Photoshop, Microsoft Windows Explorer, Microsoft WordPad, and Microsoft Media Player. It reflects 30 minutes of human activity and contains more than 200,000 OS interrupts. In the rest of the section, we refer to it as a heavy workload trace.

Our goal is to maximize the residency of the package subject to latency constraints. This is a constrained optimization problem, where the residency and latency of the package between two consecutive OS interrupts represent instantaneous rewards $r_t(\theta_t)$ and costs $c_t(\theta_t)$, respectively. The variable θ_t denotes the timeout parameter of the package PM module at the time step t. Our latency constraints are averages over 10 second periods, which corresponds to $\tau = 640$. The purpose of the constraints is to restrict the rate of bad PM actions over longer periods of time, which may affect the performance of

MM05 trace	Residency [%]		Residency regret [%]		Constraint violation [%]		
Latency budget c_0	Lazy learner	Best expert	Lazy learner	Bound	Lazy learner	Bound $U\tau$	Bound $(T/L)\tau$
0.02	74.78	74.89	0.11	8.36	0.00	9.30	85.22
0.04	77.93	77.87	−0.06	8.36	0.05	35.48	85.22
0.06	78.63	78.75	0.12	8.36	0.01	39.11	85.22

Heavy workload trace	Residency [%]		Residency regret [%]		Constraint violation [%]		
Latency budget c_0	Lazy learner	Best expert	Lazy learner	Bound	Lazy learner	Bound $U\tau$	Bound $(T/L)\tau$
0.02	24.38	24.77	0.39	10.33	0.65	5.28	130.08
0.04	29.77	29.79	0.02	10.33	0.98	60.73	130.08
0.06	32.50	32.68	0.18	10.33	0.23	11.62	130.08

Figure 5: Evaluation of our bounds on the quality of lazily-learned policies. The experimental setup is identical to Figure 4.

the computer. We experiment with different latency budgets c_0 to show the generality of our solution.

All online solutions to our optimization problem are computed by the lazy learner (Figure 2). The experts $\xi_1, \ldots, \xi_N$ are adaptive timeout policies, which adapt their timeout parameters $\xi_n(t)$ by the standard weighted majority algorithm (Kveton *et al.* 2007). Their loss function is defined as:

$$-r_t(\xi_n(t)) + \lambda_n c_t(\xi_n(t)), \tag{11}$$

where the parameter λ_n reflects the preference of the expert ξ_n for maximizing rewards and incurring costs. Our pool of experts involves six policies, which are parameterized by the following values λ_n:

$$\begin{array}{lll} \lambda_1 = 0 & \lambda_3 = 4 & \lambda_5 = 8 \\ \lambda_2 = 2 & \lambda_4 = 6 & \lambda_6 = 10. \end{array} \tag{12}$$

The values represent the aggressiveness by which the experts switch to a conservative timeout parameter. The policies are arbitrated to satisfy all latency constraints. Since the policies perform significantly better than state-of-the-art solutions to package PM (Kveton *et al.* 2007), we use them as baselines for evaluating the quality of lazily-learned policies.

The learning rate η and the segment length L are set to the values of $T^{-\frac{3}{4}}$ and $T^{\frac{1}{2}}$, respectively. This setting guarantees that both bounds in Proposition 1 are sublinear in T. In turn, the average regret and constraint violation of the lazy learner vanish with an increasing time horizon T. Our experimental results are presented in Figures 4 and 5. Without loss of generality, we report average residency and constraint violation results, and not the total values. Our bounds in Proposition 1 are additionally normalized by the time horizon T to account for this difference.

Experimental results

Figure 4 demonstrates that our lazily-learned online policies yield almost as high residency as the best expert in hindsight. The difference in residencies is always less than 0.5 percent. The policies also perform significantly better than the worst experts, which often yield less than 5 percent residency than the best experts. On the heavy workload trace, this 5 percent absolute improvement corresponds to a 20 percent improvement when measured in relative numbers. At the same time, our policies violate less than 1 percent of all constraints (Figure 5). These results suggest that lazy learning is a practical way of solving constrained optimization problems online.

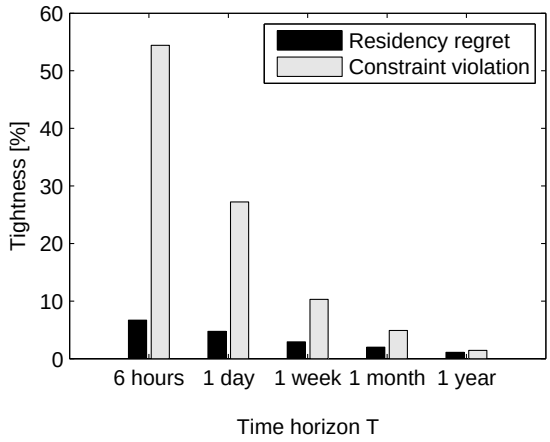

Figure 6: Tightness of our bounds with respect to an increasing time horizon T.

Figure 5 illustrates that both the regret and constraint violation of our online policies are within the bounds suggested by Proposition 1. Unfortunately, the bounds are rather loose to explain the performance of the policies. The reason is that the bounds hold against an arbitrarily adverse opponent. Our CPU traces are definitely not generated in such a fashion.

Nevertheless, note that our bounds get tighter with longer time horizons T (Figure 6). For instance, if our CPU activity traces were 1 year long, the residency regret bound would be 1.1 percent, and the constraint violation bound would be 1.4 percent. These theoretical guarantees would be sufficient for practical purposes.

Conclusions

Although online learning has been studied extensively by the machine learning community, solving constrained optimization problems online remains a challenging problem. In this paper, we proposed a practical online solution to constrained optimization problems with average-cost constraints. Moreover, we provided guarantees on its regret and the amount of constraint violations. Finally, we evaluated the solution on a challenging real-world problem. Results of our experiments show that online learning with constraints can be carried out successfully in practice.

Results of this paper can be extended in several directions. First, our online learning solution generalizes to constrained optimization problems, which involve multiple average-cost constraints at every time step t. An interesting open question is how the quality of arbitration deteriorates in such a setting. Second, arbitration is by no means the most efficient way of guaranteeing that online learned policies violate only a small number of constraints. Since arbitration typically yields zero rewards, it is important to develop online learning solutions that do not require it. Note that the work of Mannor and Tsitsiklis (2006) implies that it is generally infeasible to achieve close-to-optimal rewards without violating constraints.

Acknowledgment

We thank anonymous reviewers for comments that led to the improvement of this paper.

References

Bertsekas, D. 1999. *Nonlinear Programming*. Belmont, MA: Athena Scientific.

Cesa-Bianchi, N., and Lugosi, G. 2006. *Prediction, Learning, and Games*. New York, NY: Cambridge University Press.

Dhiman, G., and Simunic, T. 2006. Dynamic power management using machine learning. In *Proceedings of the 2006 IEEE / ACM International Conference on Computer-Aided Design*.

Gramacy, R.; Warmuth, M.; Brandt, S.; and Ari, I. 2003. Adaptive caching by refetching. In *Advances in Neural Information Processing Systems 15*, 1465–1472.

Helmbold, D.; Long, D.; Sconyers, T.; and Sherrod, B. 2000. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications* 5(4):285–297.

Karlin, A.; Manasse, M.; McGeoch, L.; and Owicki, S. 1994. Competitive randomized algorithms for nonuniform problems. *Algorithmica* 11(6):542–571.

Kveton, B.; Gandhi, P.; Theocharous, G.; Mannor, S.; Rosario, B.; and Shah, N. 2007. Adaptive timeout policies for fast fine-grained power management. In *Proceedings of the 19th Conference on Innovative Applications of Artificial Intelligence*.

Littlestone, N., and Warmuth, M. 1994. The weighted majority algorithm. *Information and Computation* 108(2):212–261.

Mannor, S., and Tsitsiklis, J. 2006. Online learning with constraints. In *Proceedings of 19th Annual Conference on Learning Theory*, 529–543.

Zinkevich, M. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, 928–936.