Online Convex Optimization for Online Job Scheduling

Zhibo Yang
Department of Computer Science
Stony Brook University
zhibyang@cs.stonybrook.edu

1 Introduction

Job scheduling/resource allocation problem has been a classical genre of problem in many fields for decades. In this problem, usually each job requires some certain amount of resource and also may have certain constraints (e.g., time). The goal is usually find a schedule for these jobs that optimizes some objective function (e.g., maximize overall system throughput and minimize the overall flow time of all jobs). Toward this end, researchers have developed many types of algorithms to solve it, e.g., greedy algorithm. But most of these algorithms are designed under a offline setting—information about these jobs are known beforehand. However, reality is often the opposite—jobs keep coming in the system and the decision needs to be made online. Therefore, in this project, I will work toward this online job scheduling problem and using relatively new technique, called Online Convex Optimization (OCO), to solve it and see how it performs by comparing it with classical greedy algorithm. Experimental results show that OCO approach achieves better performance than the shortest-job-first method and reduces the overall flow time by 33.3% in my experiments. The source code is released at https://github.com/ouyangzhibo/CSE592-Online-Convex-Optimization.

2 A Brief Overview of OCO

Under the setting of OCO, we assume that we are making a sequence of decision against the nature: At each time slot $t$, we make a decision or an action $x_t$ by choosing from a known convex set $X$. The nature then will reveal a loss function $f_t(\cdot)$ which is convex and we suffer a $f_t(x_t)$ from the decision $x_t$ we made. Notice that we can access the loss function $f_t(\cdot)$ only after we made the decision $x_t$ at time $t$. The goal here is to find a sequence of $\{x_t\}$ that minimizes the accumulative loss till time $T$:

$$\sum_{t=1}^{T} f_t(x_t).$$

A static regret [3] is defined to measure the performance of an OCO algorithm as follows:

$$\text{Reg}_T^s = \sum_{t=1}^{T} f_t(x_t) - \min_{x \in X} \sum_{t=1}^{T} f_t(x),$$

which measures the different between the loss of $\{x_t\}$ and the best static decision in hindsight. Many well-known algorithms for convex optimization are extended to an online version to solve the OCO problem [1], e.g., online gradient descent and online mirror descent can achieve a bound of $O(\sqrt{T})$ on the regret, which means the average regret $\text{Reg}_T^s / T \to 0$ as $T$ approaches $+\infty$.

But recent studies [5] show that static regret is a rather coarse metric for benchmarking OCO algorithms. Hence, as oppose to static regret, dynamic regret [5] is recently introduced as a more
competitive metric for OCO algorithms

\[
\text{Reg}^d_T = \sum_{t=1}^{T} f_t(x_t) - \min_{\{x_t \in X, \forall t\}} \sum_{t=1}^{T} f_t(x_t).
\]

Unlike static regret, dynamic regret is formed by the best possible dynamic decisions with loss function known beforehand. Obviously,

\[
\text{Reg}^d_T \geq \text{Reg}^s_T,
\]

since it is easy to see that the accumulative loss of a static predictor must be larger than that of a dynamic predictor.

3 Problem Formulation

Consider a computer cluster consisting of multiple computers, each of which represents a collection of resources (e.g., CPU cores and memory). We have a time-slotted system where job \( j \) arrives at time slot \( t \), denoted by \( a_j \). Upon arrival, job \( j \) will be stored in some global queuing system and wait to be scheduled at some future some time slot. Each job \( j \) requires certain units of resources for a certain amount of time before its completion. Denote by \( c_j \) and \( p_j \) the completion time and size (duration) of job \( j \) and \( d_t \) the units of work to be done for all the jobs at time \( t \). The flow time of job \( j \) is the difference between its completion time and arrival time, i.e., \( c_j - a_j \). In this project, I simplify the system a little bit by considering only one type of resource, CPU, and the whole cluster has one unit of CPU to serve the jobs for every time slot. Aimed at finding a sequence of schedule of jobs/allocation of resources \( \{x_t\} \) where \( x_{jt} \in \{0, 1\} \) that minimizes the overall flow time of all the jobs given \( N \) jobs, I formulate it as the following optimization problem:

\[
\min_{\{x_t\}} \sum_{j=1}^{N} c_j - a_j \\
\text{s.t.} \quad d^T x_t \leq 1, \quad t = 1, 2, \cdots, T \\
0 \leq x_t \leq 1, \quad t = 1, 2, \cdots, T \\
x_t \geq d_t, \quad t = 1, 2, \cdots, T.
\]

The first set of constraints are cluster capacity constraints meaning that the overall resource allocated cannot exceed its capacity. The second set of constraints is convex relaxation of the discrete decision variables \( \{x_t\} \), which make the problem easier to solve. The last set of constraints are the job demand constraints, i.e., the amount of resource allocated to each jobs should at least satisfy its demand.

However, the objective of (1) is non-convex and very hard to solve in general. Im et. al. [4] propose to approximate the \( \ell_k \) norm of the \( (c_j - a_j) \) vector, \( \sum (c_j - a_j)^k \), using a convex function of \( \{x_t\} \) as follows

\[
\sum_{t,j} \left( \frac{(t-a_j)^k}{p_j} + p_j^{k-1} \right) x^j_t,
\]

where \( k = 1 \) in our case. (2) is the so-called fractional flow time. Hence, instead of solving the difficult non-convex problem (1), we solve a approximated alternative convex optimization problem as follows

\[
\min_{\{x_t\}} \sum_{t,j} \left( \frac{(t-a_j)^k}{p_j} + p_j^{k-1} \right) x^j_t \\
\text{s.t.} \quad d^T x_t \leq 1, \quad t = 1, 2, \cdots, T \\
0 \leq x_t \leq 1, \quad t = 1, 2, \cdots, T \\
x_t \geq d_t, \quad t = 1, 2, \cdots, T.
\]

(3) is shown to be 2-approximation of the original problem (1). Namely, the optimum of (3) \( OPT2 \) is no greater than two times of the optimum of (1) \( OPT1 \), i.e.,

\[
OPT2 \leq 2 \times OPT1
\]
4 Solving OCO

There are a handful of OCO algorithms, e.g., online (sub)gradient descent (OGD), online mirror descent (OMD), see more details at [1]. But most of these algorithms can only achieve a sub-linear bound on static regret, while the average dynamic regret can go unbound. Therefore, to address this problem, I adopted a online primal-dual algorithm – Modified Online Saddle Point (MOSP) method which is shown to have a sub-linear dynamic regret under some mild assumptions [3]. Moreover, MOSP allows the time-varying constraints $g_t(x_t) \leq 0$ to be violated but satisfied in the long run, i.e., $\sum_t g_t(x_t) \leq 0$. Hence, we are able to impose another set of practical constraints, that is, the processing rate of each job $j$ at each time slot $t$ should be no smaller than its average processing speed, i.e.,

$$x^j_t(t) \geq \frac{p_j}{T-a_j}, \quad \forall j, t.$$  

Because otherwise by the end of the time the jobs would be uncompleted. Algorithm 1 presents the MOSP algorithm.

**Algorithm 1: Modified Online Saddle Point Algorithm [3]**

**Initialize:** primal solution $x_0$, dual solution $\lambda_1$, and positive stepizes $\alpha$ and $\mu$.

**for** $t = 1, 2, \cdots$ **do**

**Update primal $x_t$:**

$$x_t = \arg \min_{x \in \mathcal{X}} \nabla f_{t-1}(x_{t-1})(x - x_{t-1}) + \lambda^T_{t-1} g_{t-1}(x) + \frac{||x - x_{t-1}||^2}{2\alpha}.$$  

Observe loss $f_t(x_t)$ and constraint $g_t(x_t)$.

**Update dual $\lambda_{t+1}$:**

$$\lambda_{t+1} = \max(0, \lambda_t + \mu g_t(x_t)).$$  

5 Experimental Results

To evaluate the performance of applying OCO algorithm in the application of online job scheduling, I compare the OCO algorithm (specifically, MOSP) with the classical shortest-job-first method under simulated environments. In my simulation, there are 100 jobs randomly coming in a time-slotted system of 150 time slots in total. The cluster has 10 CPUs to be scheduled in each time slot. Each job requires a random amount of CPUs varying from 1 to 10 for a random duration ranging from 0 to 3 with 0.5 as step size.

![Figure 1: Comparison between OCO approach with the shortest-job-first baseline.](image-url)
Figure 1 represents the experimental results. In particular, I show the overall flow time and fractional flow time of all the jobs in the system for each time slot. It is easy to observe that the OCO approach outputs a better schedule of much less flow time than the shortest-job-first scheme, reducing it by 33.3%. Moreover, by comparing the fractional flow time and flow time of OCO, we can see that fractional flow time is very close to the flow time, meaning the fractional flow time (3) serves as a good approximation of our original objective (1). However, the OCO approach does have a drawback compared to the greedy baseline, i.e., OCO approach requires more computation because at each time slot OCO algorithm needs to solve a convex optimization problem.

6 Conclusion and Acknowledgement

In this project, I applied online convex optimization approach in solving online job scheduling problem. The experimental results showed that OCO approach performs better than shortest-job-first method at the cost of more computational power. In this general, applying online convex optimization in online resource scheduling (e.g., online job scheduling) is a promising area of study.

In the end, I would like to thank Francesco Orabona for advising this work and for teaching the course CSE 592 (convex optimization) where I really had fun and learned a lot.

References


