Online Optimization of Smoothed Piecewise Constant Functions

January 26, 2017

Abstract

We study online optimization of *smoothed* piecewise constant functions over the domain $[0, 1)$. This is motivated by the problem of adaptively picking parameters of learning algorithms as in the recently introduced framework by Gupta and Roughgarden (2016). Majority of the machine learning literature has focused on Lipschitz-continuous functions or functions with bounded gradients.[1] This is with good reason—any learning algorithm suffers linear regret even against piecewise constant functions that are chosen adversarially, arguably the simplest of non-Lipschitz continuous functions. The *smoothed* setting we consider is inspired by the seminal work of Spielman and Teng (2004) and the recent work of Gupta and Roughgarden (2016)—in this setting, the sequence of functions may be chosen by an adversary, however, with some uncertainty in the location of discontinuities. We give algorithms that achieve sublinear regret in the full information and bandit settings.

1 Introduction

In this paper, we study the problem of online optimization of piecewise constant functions. This is motivated by the question of selecting *optimal parameters* for learning algorithms. Recently, Gupta and Roughgarden (2016) introduced a *probably approximately correct* (PAC) framework for choosing parameters of algorithms. Imagine a situation, when a website wishes to provide personalized results to a user. To respond to a user's query, the service provider may need to implement a learning (or some other type of) algorithm which involves choosing parameters. The choice of parameters affects the quality of solution and ideally we would like to design a mechanism where the service provider learns from past instances, or at least employs a strategy that has low regret with respect to the *single* optimal solution in hindsight. In many learning problems, the goal is to find parameters by optimizing a continuous function (of the parameters); however, ever so often one encounters problems with discrete solutions, such as k-means or independent set, which result in objective functions that have discontinuities.

Concretely, we consider the problem of online optimization of piecewise constant functions over the domain $[0, 1)$. At each round the learning algorithm plays a point $x_t \in [0, 1)$, receives payoff $f_t(x_t)$, where f_t is a piecewise constant function. The aim of the learning algorithm is to achieve *no-regret* with respect to the best single point $x^* \in [0, 1)$ in hindsight.[2] As is standard, by *no regret* we mean, regret that grows sub-linearly with T, the number of rounds played. If we make no assumptions about how the piecewise linear functions f_t are chosen then, it is easy to construct instances where the algorithm would suffer regret that is linear in T.

[1] These functions are typically called *smooth* in the machine learning literature. We avoid this usage here, since we use *smoothed* and *smoothness* in the sense of Spielman and Teng (2004).

[2] Unfortunately, the confusing terminology *no-regret* is common in the literature and is used to denote that the regret grows sublinearly in T, the number of rounds for which the online algorithm is run.

We take the view that real-world problems, while not entirely stochastic are rarely truly adversarial.[3] In this work, we consider a *smoothed* adversary; rather than defining a piecewise constant function f over $[0, 1)$ by defining the intervals $[0, a_1), \ldots [a_{k-1}, 1)$ exactly, the adversary may only define distributions to pick the points a_i, with the added constraint that the density of these distributions is upper bounded by some parameter σ. It is very natural to assume that there is uncertainty in defining real-world problems, either due to noise or imperfect information; indeed this was also the motivation of the original work by Spielman and Teng (2004) where they showed that the smoothed time complexity of (a variant of) the simplex algorithm is polynomial. This uncertainty in defining the intervals (or the points of discontinuity) is precisely what we exploit in designing no-regret algorithms.

Machine learning research has primarily focused on optimizing functions with bounded (first few) derivatives. For example, there exists substantial literature on online optimization of Lipschitz continuous functions, both in the full information and the bandit setting (see *e.g.,* (Kleinberg, 2004; Kleinberg et al., 2008; Bubeck et al., 2009)). However, any sort of combinatorial structure typically introduces discontinuities in the objective function. Thus, most of the existing methods for online optimization are no longer applicable.

The *smoothness* formulation we use in the paper restricts an adversary from being able to define too narrow an interval in which *optimal solutions* may lie. In particular, if we consider the refinement of all the intervals we get over T rounds of the (smoothed) adversary choosing piecewise constant functions, the smallest interval is still polynomially small in T (and the bound σ on the density and the number of pieces k). This ensures that the problem is not that of finding a needle in a haystack, but a rather hefty iron rod. Under these conditions, in principle, one could simply draw a large enough (but still polynomial in T) number of points uniformly in the interval $[0, 1)$, and consider the problem as the standard experts setting. The bandit setting is a bit more delicate, but still could be handled using ideas similar to the Exp.4 algorithm (Auer et al., 2003). The difficulty is *computational*—we would rather design algorithms that at time step t, run in time polynomial in $\log(t)$ and other problem-dependent factors, than in time polynomial in t. With carefully designed algorithms and data-structures, we can indeed achieve this goal. We summarize our contributions below, describe related work, and then discuss how our work fits in a broader context.

1.1 Our Results

We show that against a *smoothed* adversary, one can design algorithms that achieve the almost optimal regret of $\tilde{O}(\sqrt{T})$ (the $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors) in the expert setting, *i.e.,* when we observe the entire function f_t at the end of the round. Our algorithm is based on a continuous version of the exponentially-weighted forecaster. A naïve implementation of the algorithm we propose would result in a running time that grows polynomially in t. We design a data structure based on *interval trees* or *red-black trees* (see Section 3.3 for the full description) that allows us to implement our algorithm extremely efficiently—the running time of our algorithm is $O(k \log(kt))$ at any given timestep, where k is the number of pieces of the piecewise constant function at each round. We remark that at least logarithmic dependence in t is required in the sense that even keeping track of time requires time at least $\log(t)$.

We also consider the *bandit setting*: here the algorithm does not observe the entire function f_t, but only the value $f_t(x_t)$ for the point chosen. In order to estimate the function f_t elsewhere, we optimistically assume that it is constant in some (suitably chosen) small interval around x_t. Of course, sometimes the point x_t chosen by the algorithm may lie very close to a point

[3] There may be settings where assumption of a malicious adversary is justified, *e.g.,* when spammers, google bombers, *etc.* are actively seeking to compromise systems.

of discontinuity of the function f_t. However, this cannot happen very often because of the *smoothness* constraint on the adversary. With a somewhat delicate analysis we obtain a regret bound of $\tilde{O}(\text{poly}(k, \sigma)T^{2/3})$ in the bandit setting. As in the full information (or expert) setting, our algorithm is very efficient, *i.e.,* polynomial in $\log(t)$ and other parameters such as k and σ.

In Section 5, we consider a few different problems in this setting—knapsack, weighted independent set, and weighted k-means. The first two of these problems were already considered by Gupta and Roughgarden (2016), however they do not provide *true* regret bounds in their paper, *i.e.,* bounds where the average regret approaches 0 as $T \to \infty$.[4] The combinatorial nature of these problems means that as we vary the algorithm parameters, there may be discontinuities in the objective function. Our preliminary experiments indicate that the behavior does indeed correspond to that predicted by theoretical bounds, albeit with slightly better rates of convergence (possibly since we generate instances randomly).

1.2 Related Work

The work most closely related to ours is that of Gupta and Roughgarden (2016). In the context of online learning, our work improves upon theirs in providing bounds for online learning of algorithm parameters that are *true regret* bounds; in their paper they only provide ϵ-regret bounds, in that one can guarantee that for any given ϵ the algorithm will achieve average regret of ϵ. The algorithms we present in this paper are more natural and achieve a significant improvement in running time. We also give results in the *bandit* setting, which is in many ways more appropriate for the applications under consideration. The approach considered in their paper does not yield a bandit algorithm. With some effort, one may be able to adapt ideas from the Exp.4 algorithm of Auer et al. (2003) to achieve a non-trivial regret bound in the *bandit* case; however, the resulting algorithm would be computationally expensive.

There is a substantial body of work that seeks to use learning mechanisms to choose the parameters or *hyperparameters* of algorithms. Snoek et al. (2014, 2012) suggest using Bayesian optimization techniques to choose hyperparameters effectively. Yet other papers (see *e.g.,* (Fink, 1998; Huang et al., 2010; Kotthoff et al., 2012; Hutter et al., 2015)) suggest various techniques to choose parameters for algorithms (not necessarily in the context of learning). However, except for the work of Gupta and Roughgarden (2016), most work is not theoretical in nature.

1.3 Discussion

The notion of *smoothness* considered in this paper is inspired by the seminal work of Spielman and Teng (2004). In theoretical computer science, this notion allows us to look beyond worst-case analysis without making extremely strong assumptions required for average-case analyses. This approach seems particularly relevant to machine learning, a filed in which worst-case results or those using strong distributional assumptions typically have little bearing in practice. The smoothness considered here is on the instances themselves rather than on the functions being optimized as is common in machine learning. Combinatorial problems arise naturally in several online and offline learning settings; in these cases the notion of *smoothness* à la Spielman and Teng may be more appropriate than the traditional notion of Lipschitz continuity. It would be interesting to explore if this notion of smoothness is applicable in other settings, *e.g.,* sleeping combinatorial experts and bandit settings—where it is known that *stochastic instances* are often tractable, while adversarial ones are *computationally hard* (see *e.g.,* (Kanade and Steinke, 2014; Neu and Valko, 2014; Kale et al., 2015)).

[4] We remark that their algorithm will be a "true" no-regret algorithm with suitably chosen parameters, but in that regime the running time is polynomial in T.

A natural open question is whether our work could be extended to more general functions with discontinuities, such as piecewise linear or piecewise Lipschitz functions. If computational cost were not a concern, we believe this could be achieved (at least in the full-information setting) by choosing a fine enough grid of $[0, 1)$ as experts. However, whether efficient algorithms such as ours for the case of piecewise constant functions can be designed is an open question.

2 Setting

We consider the online optimization setting where the decision space is $[0, 1)$. At each time step t, the learning algorithm must pick a point $x_t \in [0, 1)$ to play. A *smoothed oblivious adversary* picks a function $f_t : [0, 1) \to [0, 1]$ that is piecewise constant as follows:

1. Adversary defines distributions $D_{t,1}, \ldots, D_{t,k-1}$ where the support of each $D_{t,i}$ is contained in $(0, 1)$ and the density functions are bounded by σ.
2. Adversary defines values $v_{t,1}, \ldots, v_{t,k} \in [0, 1]$
3. Nature draws $a'_{t,i} \sim D_{t,i}$ independently for $i = 1, \ldots, k-1$. Let $0 = a_{t,0}, a_{t,1}, \ldots, a_{t,k-1}, a_{t,k} = 1$ be in non-decreasing order, where $a_{t,1}, \ldots, a_{t,k-1}$ are just $a'_{t,1}, \ldots, a'_{t,k-1}$ sorted.[5]
4. The piecewise constant function f_t is defined as $f_t(x) = v_{t,i}$ for $x \in [a_{t,i-1}, a_{t,i})$.

For a known time horizon T, let $x_1, \ldots, x_T$ be the choices made by the learning algorithm.[6] Then the regret is defined as:

$$\text{Regret(Alg)} = \max_{x \in [0,1)} \sum_{t=1}^{T} f_t(x) - \sum_{t=1}^{T} f_t(x_t)$$

We consider the full information (or *experts*) setting, where at the end of each round the full function f_t is revealed to the learning algorithm. We also look at the *bandit* setting, where the learning algorithm only sees the value $f_t(x_t)$. All results in this paper are stated in terms of expected regret; we believe that bounds that hold with high probability can be obtained using standard techniques.

All the results in this paper can easily be generalized to the setting where the decision space is $[0, 1]^d$ and the adversary chooses functions that are constant on sub-hypercubes. In the full-information setting, the regret guarantees will be worse by a factor that is polynomial in d and the running time worse by a factor exponential in d. In the bandit setting, both the regret and the running time will be worse by a factor exponential in d. Without further assumptions, these results are the best one can hope for as a 2^d arm bandit problem can be reduced to this problem. For simplicity we only discuss the one dimensional setting and defer the general case to the full version of the paper.

We first state some useful observation that we will use repeatedly in this paper. These are by no means original and already appear in some form in (Gupta and Roughgarden, 2016) for example.

Observation 2.1. *Let $x_1, \ldots, x_m$ be independently drawn from any distributions (possibly different) whose density functions are bounded by σ. Then the probability that there exist $x_i < x_j$ such that $x_j - x_i < \epsilon$ is at most $m^2\sigma\epsilon$.*

[5] Under the smoothness assumptions, the $a'_{t,i}$ will be distinct with probability 1.

[6] We assume that the time horizon T is known, otherwise, the standard doubling trick may be applied.

Proof. The proof is just an application of the union bound over all $\binom{m}{2}$ pairs. □

Observation 2.2. *Let $0 = x_0 < x_1 < \cdots < x_m = 1$ be any points in $[0,1]$ such that $x_i - x_{i-1} > \epsilon$ for all i. Let $y_1, \ldots, y_N$ be drawn uniformly at random from $[0,1)$. Then if $N \geq \frac{1}{\epsilon}(\ln(\epsilon^{-1}) + \ln(\delta^{-1}))$, the probability that there exists i such that there is no y_j in the interval (x_{i-1}, x_i) is at most δ.*

Proof. This is basically a balls into bins argument with at most ϵ^{-1} bins and the probability of throwing a ball into each bin is at least ϵ. □

3 Full Information Setting

First in Section 3.1, we explain why it is necessary to look at *smoothed adversaries*; without this, one can easily construct a relatively benign instance that suffers a regret of $\Omega(T)$. Then, we give an exponentially-weighted forecaster that achieves expected regret that is $O(\sqrt{\log(T)T})$. We show that in fact this can be implemented very efficiently; at time t the running time of our algorithm is polynomial in k and $\log(t)$, where k is the number of pieces of the functions.

3.1 Lower Bound for Worst-Case Adversaries

Here, we show that unless one adds a smoothness assumption the worst-case regret bound is linear in T. This is unsurprising and in a way already follows from Theorem 4.2 in the extended version of (Gupta and Roughgarden, 2016). However, in the more general setting considered in this paper, the bad instance is simpler to describe and we provide it for completeness.

We describe a simple adversary that chooses functions that are piecewise constant with at most 3 pieces and each piece being of length at least $1/5$; but for allowing the adversary to choose the points of discontinuity arbitrarily, the freedom given to the adversary is limited. For round 1 the adversary chooses f_1 such that $f_1(x) = 1$ for $x \in [2/5, 3/5)$ and 0 otherwise. On round 2 the adversary chooses f_2 to be 1 on either the interval $[3/10, 1/2)$ or $[1/2, 7/10)$ uniformly at random. Thus, any learning algorithm can get expected payoff at most $1/2$, but either the entire interval $[2/5, 1/2)$ or $[1/2, 3/5)$ would have got payoff of 1 on both rounds. In general if after t rounds, if there is an interval $[l_t, u_t)$ such that $f_s(x) = 1$ for all $x \in [l_t, u_t)$ for all $s \leq t$, then if $m_t = (l_t + u_t)/2$, the adversary picks f_{t+1} to take value on either $[m_t - 1/5, m_t)$ or $[m_t, m_t + 1/5)$. It is clear that after T rounds the expected payoff of any learning algorithm is at most $T/2$, however there exists a point $x^* \in [0,1)$ that would receive a payoff of T.

3.2 No-Regret Algorithm for Smoothed Adversaries

Algorithm 1 is a fairly standard exponentially-weighted forecaster that is used to make predictions in the non-stochastic setting. We describe the efficient implementation using *modified* interval trees in Section 3.3. If one were merely concerned with achieving a running time that was polynomial in T at each round, there is a rather easy solution. Using Observation 2.1 we know that even after a full refinement of intervals that appear in the functions $f_1, \ldots, f_T$ with high probability the smallest interval is of length at least $1/T^3$. Then Observation 2.2 tells us that by picking $T^3 \log(T)$ points uniformly at random, we get a set of points that would hit every interval in the refinement. Thus, we could pick these points to begin with and apply a standard expert algorithm, such as Hedge (Freund and Schapire, 1995). Since the regret depends only logarithmically on the number of experts, we would still achieve a regret bound of $O(\sqrt{\log(T)T})$. In a way, this is what Gupta and Roughgarden (2016) do to obtain their "low-regret" bound. However, this solution is inelegant and suffers significantly in terms of

computational cost. Our algorithm runs in time polynomial in k and $\log(t)$ at time t; note that dependence on $\log(t)$ is required even just to keep track of time!

Algorithm 1 No-regret algorithm

input: η

set $F_1(x) = 0$ to be the constant 0 function over $[0, 1)$

for $t = 1, 2, \ldots, T$ **do**

1. Define $p_t(x) = \frac{\exp(\eta F_t(x))}{\int_0^1 \exp(\eta F_t(x))dx}$
2. Pick $x_t \sim p_t$
3. Observe function f_t and receive payoff $f_t(x_t)$
4. Set $F_{t+1} = F_t + f_t$

Theorem 3.1. *With probability* 1*, the expected (the expectation is only with respect to the random choices of the algorithm, but not those of nature) regret of Algorithm 1 is bounded by*

$$\eta(e-2)T - \frac{\ln(\epsilon^*)}{\eta}$$

where if $x^* \in [0, 1)$ is the best point in hindsight such that $x^* \neq a_{t,j}$ for any $t \in [T]$ and $j \in [k]$, i.e., x^* is not a point of discontinuity for any of the functions f_t,

$$\epsilon^* = \min\{a_{t,j} \mid t \in [T], j \in [k], a_{t,j} > x^*\} - \max\{a_{t,j} \mid t \in [T], j \in [k], a_{t,j} < x^*\},$$

Here ϵ^* is the length of the largest interval containing the point x^* for which F_{T+1} is constant (and optimal).

Proof. Let $x^* \in \operatorname{argmin} F_{T+1}(x)$ such that $x^* \notin \{a_{t,j} \mid t \in [T], j \in [k]\}$. Note that such an x^* exists as long as all the functions have distinct points of discontinuity, which happens with probability 1 under our model of max density assumption. Also let $a^* = \min\{a_{t,j} \mid a_{t,j} > x^*\}$ and $b^* = \max\{a_{t,j} \mid a_{t,j} < x^*\}$. Then, note that F_{T+1} is constant on the interval (a^*, b^*).

Let $W_t = \int_0^1 \exp(\eta F_t(x))dx$. We follow the fairly standard proof of obtaining regret bounds for exponentially-weighted forecasting. Let $P_t = \mathbb{E}_{x \sim p_t}[f_t(x)]$ denote the expected payoff achieved by the algorithm in round t, where the expectation is only with respect to the algorithm's random choices. The first observation is to upper bound W_{t+1}/W_t by $\exp((e^\eta - 1)P_t)$. To see this,

$$\begin{aligned}
\frac{W_{t+1}}{W_t} &= \frac{\int_0^1 \exp(\eta(F_{t+1}(x))dx}{\int_0^1 \exp(\eta(F_t(x))dx} \\
&= \frac{\int_0^1 \exp(\eta F_t(x))e^{\eta f_t(x)}dx}{\int_0^1 \exp(\eta(F_t(x))dx} && \text{Since } F_{t+1} = F_t + f_t \\
&= \int_0^1 p_t(x)e^{\eta f_t(x)}dx && \text{By definition of } p_t \\
&\leq \int_0^1 p_t(x)(1 + (e^\eta - 1)f_t(x))dx && \text{For } z \in [0,1],\ e^{\eta z} \leq 1 + (e^\eta - 1)z \\
&\leq 1 + (e^\eta - 1)P_t \leq \exp((e^\eta - 1)P_t) && \text{Since } 1 + z \leq e^z
\end{aligned}$$

Thus, we get that

$$\frac{W_{T+1}}{W_1} \leq \exp\left((e^\eta - 1)\sum_{t=1}^{T} P_t\right) = \exp((e^\eta - 1)P(\text{Alg})) \tag{1}$$

Where $P(\text{Alg}) = \sum_{t=1}^{T} P_t$ is the expected payoff of the algorithm (with respect to its random choices).

On the other hand $W_1 = \int_0^1 \exp(\eta F_1(x))dx = 1$ and,

$$\begin{aligned} W_{T+1} &= \int_0^1 \exp(\eta F_{T+1}(x))dx && (2)\\ &\geq \int_{a^*}^{b^*} \exp(\eta F_{T+1}(x))dx \qquad \text{Since } \exp(\eta F_{T+1}(x)) \geq 0 \text{ on } [0,1) && (3)\\ &= \epsilon^* \exp(\eta \text{OPT}) \qquad \text{where OPT} = F_{T+1}(x^*) && (4) \end{aligned}$$

Thence, we have,

$$\frac{W_{T+1}}{W_1} \geq \epsilon^* \exp(\eta \text{OPT}) \tag{5}$$

Using (1) and (5) together, after taking logarithms of both sides, we get

$$\eta \text{OPT} + \log(\epsilon^*) \leq (e^\eta - 1)P(\text{Alg})$$

Rearranging terms and dividing by η, we get

$$\text{OPT} - P(\text{Alg}) \leq \frac{e^\eta - 1 - \eta}{\eta} P(\text{Alg}) - \frac{\log(\epsilon^*)}{\eta}$$

For $\eta \in [0,1]$, $e^\eta \leq 1 + \eta + (e-2)\eta^2$ and $P(\text{Alg}) \leq T$ since the range of f_t is $[0,1]$. Thus, we get

$$\text{OPT} - P(\text{Alg}) \leq \eta(e-2)P(\text{Alg}) - \frac{\log \epsilon^*}{\eta}$$

□

In order to obtain the appropriate regret bound we just need to apply Observation 2.1 to get a bound on the value of ϵ^*. Note that by choosing $\delta = 1/T$ and assuming that $k, \sigma < T$, we get an expected regret bound of the form $O(\sqrt{\log(T)T})$, where the expectation is taken with respect to the random choices of both the algorithm and nature.

Corollary 3.2. *The expected regret of Algorithm 1 is bounded by* $2\sqrt{(e-2)\log(k^2T^3\sigma)T} + 1$. *The expectation is with respect to the random choices of the algorithm as well as nature.*

Proof. Using Observation 2.1, we know that with except with probability $\delta = 1/T$, $\epsilon^* \geq 1/(k^2T^3\sigma)$; also $P(\text{Alg}) \leq T$ (for any choices of nature). Substituting these two bounds in the statement of Theorem 3.1 and setting $\eta = \sqrt{\frac{\log(k^2T^3\sigma)}{(e-2)T}}$ gives the required bound. The $+1$ term accounts for the fact that with probability $\delta = 1/T$, the regret could be as high as T. □

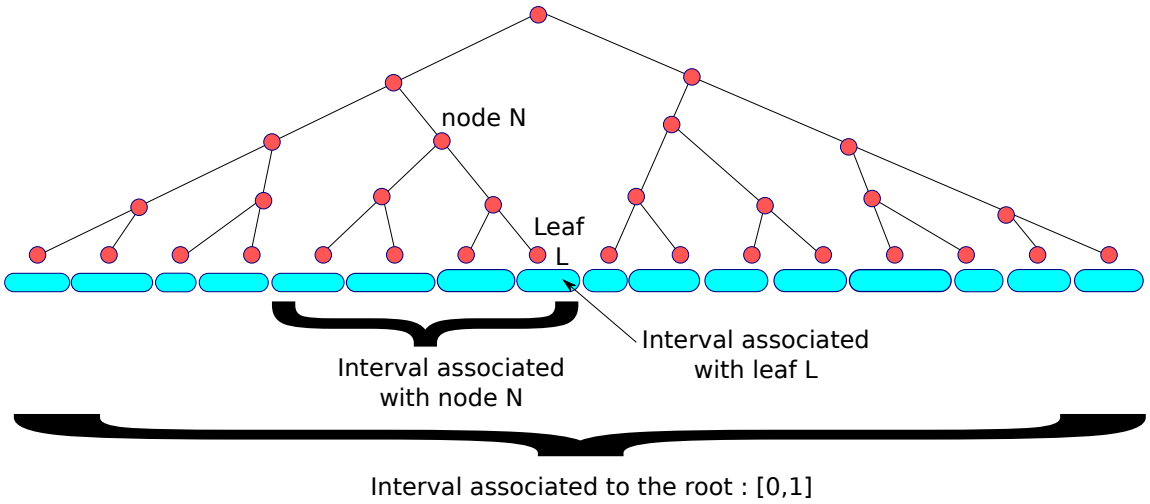


Figure 1: Example of an interval tree over the interval $[0, 1]$.

Remark 3.3. *Rather than considering a smoothed adversary, one may consider a slightly different notion of regret. For example, one can consider the best "small" enough interval, but the comparison point is chosen to be the worst point in that interval (in other words some kind of a minimax criterion). Obviously, choosing the best point is impossible as discussed in the example in Section 3.1. Our algorithm and proof technique work essentially unchanged in this setting. The only change is that in Step* (3) *we will use this interval instead. The regret bound depends on the inverse of the length of the interval, but only logarithmically.*

Running Time. A naïve implementation of Algorithm 1 would result in running time that is polynomial in t at time t (apart from also being polynomial in k and σ). This is because the number of intervals in the refinement increase linearly in T. However, by using an *augmented* interval tree data structure to store the past functions and using *messages* to perform updates lazily, we can guarantee that the running time of the algorithm is polynomial in $\log(t)$. The sampling required in Step 2 of the algorithm can also be implemented efficiently by storing auxiliary information about the *weights* of intervals. The next subsection explains this data structure in detail.

3.3 An Efficient Data-Structure

In this section, we describe a data structure that ensures that the *selection* and *update* steps (steps 2, 4) of Algorithm 1 have a running time of $O(k \log(tk))$ at time t.

We first describe the high level idea. In order to have an efficient implementation, we exploit the fact that the function is piecewise constant with k pieces. We build upon a data structure called *interval trees* (see (Cormen et al., 2009) for more details), that for any partition P of the interval $[0, 1)$ into n parts, maintains a binary tree with n leaves that has the following properties:

(i) The leaves are the parts of P.

(ii) Any level of the tree corresponds to a coarsening of the partition. More precisely, for any internal node N, there is an interval associated with N which corresponds to the interval that is the union of the intervals associated with the children of N (see Figure 1).

With such a data-structure, it is possible to refine the partition (split existing intervals), and

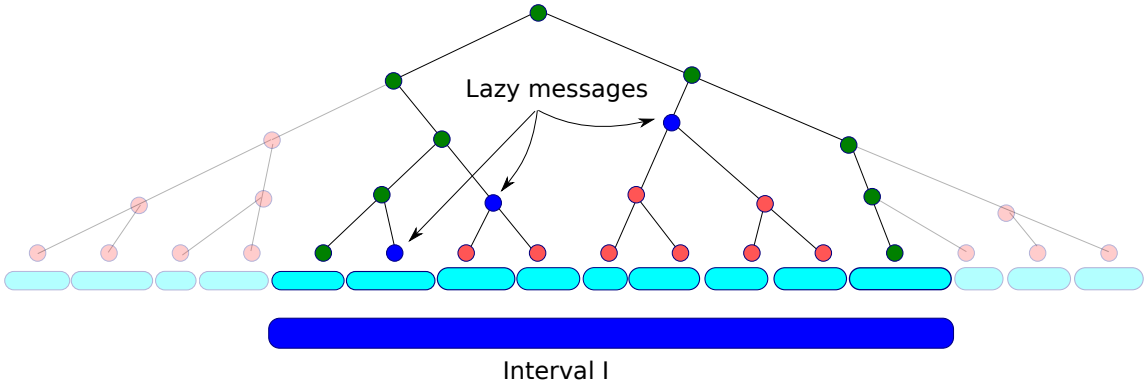


Figure 2: Example of a call to the update procedure. The interval to update is the interval I. Messages are left along the path connecting the two extremities (smallest and highest intervals) of I. In green are the nodes that get their messages updated ($m(N) = 1$ for those nodes). In blue are the nodes that get a new message corresponding to $\exp(\eta f_t(I))$ ($m(N) \leftarrow m(N) \cdot \exp(\eta f_t(I))$ for those nodes). This update should be propagated to all the descendants of the green nodes (the red nodes), but will only be done lazily when required. The lazy approach and the structure of the tree allows to bound the number of green and blue vertices by $O(\log T)$. Indeed, observe that not all the nodes of the subtrees corresponding to interval I are updated at this step.

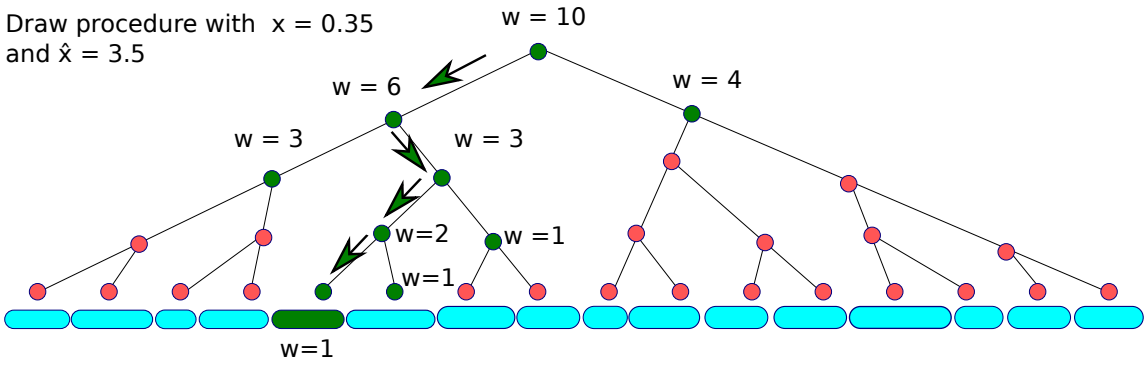


Figure 3: Example of call to the draw procedure. Assuming the value of x is 0.35, the procedure moves along the tree based on the values of the $w(N)$ to find the interval corresponding to $p_t(x)$. The vertices in green are the vertices whose pending messages are updated by the procedure (i.e.: the value of $m(N)$ for each node N is 1 after the call).

establish membership in time $O(\log n)$, where n is the number of leaves in the tree. In our setting, n will be $O(tk)$ after t steps.

Here, we extend this data structure to support additional operations. Note that F_t is piecewise constant, in particular F_t is constant over $I(\ell)$, the interval defined at leaf ℓ; by abuse of notation let $F_t(I(\ell))$ denote this constant value. For each leaf ℓ of the tree we will maintain a variable $w(\ell)$ whose value is $|I(\ell)| \exp(\eta F_t(I(\ell)))$, where $I(\ell)$ is the length of the interval corresponding to leaf ℓ. Then, for each internal node N of the tree, whose associated interval is $I = [a, b)$, we will maintain

$$w(N) = \sum_{\substack{\ell \text{ leaf of the subtree} \\ \text{rooted at } N}} |I(\ell)| \exp(\eta F_t(I(\ell))) = \int_a^b \exp(\eta F_t(x))dx.$$

This allows us to encode the cumulative distribution function of p_t defined at Step 1 of Algorithm 1. Thus, starting from the root and moving toward the leaves, it is possible to draw from this distribution in time $O(\log(kt))$ (see Fig 3 and the description of the Draw procedure).

Then, at Step 4 of Algorithm 1, we know that the function f_t is piecewise constant with k pieces. For each such *piece* I, we need to set $F_{t+1}(I') = F_t(I') + f_t(I)$ for each interval $I' \subseteq I$ on which F_t is constant. We proceed in two steps. First, we ensure that I is the disjoint union of intervals corresponding to subtrees by splitting at most two existing intervals, those that contain the endpoints of I. This operation can be implemented in interval trees of size n in time $O(\log(n))$. Then, we need to update $w(N)$ for each node N of the subtrees. Since f_t is constant on the interval I, we want to set $w(N) \leftarrow w(N) \cdot \exp(\eta f_t(I))$ for each such node N. However the number of such intervals may be $\Omega(tk)$ at time t and doing so naïvely would result in a time complexity of $\Omega(tk)$. Here we are aiming at time complexity $O(k \log(kT))$. To achieve this, we make the updates in a lazy fashion. For any interval I on which f_t is constant, we consider the leaves l and h of the tree that contain the extremities of I (recall that membership can be computed in time $O(\log n)$ for a tree of size n). We then update the values of the variables w for all the nodes along the path joining l to h. For each child of these nodes, we leave a *message*, that contains the value of the update for the subtree rooted at this child. The message at a node will be applied to the node and propagated to its children only when it is needed in the future (see Figure 2). Thanks to the structure of the tree and since f_t is piecewise constant, we show that no information is lost through this process.

We now provide a more formal description of the data structure. The data structure consists of a tree where each node N contains the following information:

- An *interval* $I(N)$.
- Possibly two *children* $l(N)$ and $h(N)$. Moreover, $I(l(N)) \cap I(h(N)) = \emptyset$ and $I(l(N)) \cup I(h(N)) = I(N)$ if they exist.
- A parent node (except for the root), $p(N)$.
- A *weight* $w(N)$. Initially, $w(N) = |I(N)|$.
- A *message* $m(N)$, initially $m(N) = 1$ ($m(N) = 1$ indicates that N is up-to-date).

We define several operations for any given tree $\mathcal{T}$.

- UpdateMessage(N): The procedure makes N and its two children, $l(N)$ and $h(N)$ up-to-date and propagate the message toward the children of $l(N)$ and $h(N)$:
  1. $w(N) \leftarrow w(N) \cdot m(N)$

2. $w(l(N)) \leftarrow w(l(N) \cdot m(l(N)) \cdot m(N)$
3. $w(h(N)) \leftarrow w(h(N) \cdot m(h(N)) \cdot m(N)$
4. For each child N' of $l(N)$: $m(N') \leftarrow m(N') \cdot m(N) \cdot m(l(N))$
5. For each child N' of $h(N)$: $m(N') \leftarrow m(N') \cdot m(N) \cdot m(h(N))$
6. $m(N) \leftarrow 1$, $m(h(N)) \leftarrow 1$, $m(l(N)) \leftarrow 1$

- Insert(I): The insertion procedure takes as input an interval $I = [l, h)$ and proceeds as the standard insertion procedure for interval trees in order to keep a balanced binary tree. More precisely, it first splits the interval $I_0 = [l_0, h_0)$ containing l into two intervals $I_0^0 = [l_0, l)$, $I_0^1[l, h_0)$ and sets $w(I_0^0) \leftarrow |I_0^0| \cdot w(I_0)/|I_0|$ and $w(I_0^1) \leftarrow |I_0^1| \cdot w(I_0)/|I_0|$. It then proceeds similarly with the interval containing h. In addition, before applying the standard insertion procedure, for every node N top-down from the root to each node that will be considered during the insertion, the procedure applies UpdateMessage. Finally, after the call to the standard insertion procedure, for each non-leaf node N considered by the procedure in a bottom-up fashion, it sets $w(N) \leftarrow w(l(N)) + w(h(N))$ and $I(N) \leftarrow I(l(N)) \cup I(h(N))$.

- Update($I = [l, h), w$): Given an interval $I = [l, h)$, let ℓ_l be the leaf whose interval starts at l and ℓ_h be the leaf whose interval ends at h. If those leaves do not exist, the procedure starts by calling Insert($[l, h)$) to create them. Let $r_{l,h}$ be the lowest common ancestor of ℓ_l and ℓ_h in $\mathcal{T}$. Let P_l be the path joining ℓ_l to $r_{l,h}$ and P_h the path joining ℓ_h to $r_{l,h}$. Then the following are executed (see Figure 2):

  1. For every node on the path from the root to $r_{l,h}$ in a top-down order, apply the UpdateMessage procedure.
  2. For every node along the path P_l, from $r_{l,h}$ to ℓ_l in a top-down order, apply the UpdateMessage procedure.
  3. For every node along the path P_h, from $r_{l,h}$ to ℓ_h in a top-down order, apply the UpdateMessage procedure.
  4. At node ℓ_l, update $w(\ell_l) \leftarrow w(\ell_l) \cdot w$. For every internal node on the path P_l, *i.e.,* all nodes except for ℓ_l and $r_{l,h}$, the following updates are made in bottom-up order (going from ℓ_l to $r_{l,h}$).
     (a) At node N, if $l(N) \in P_l$, then set $w(N) \leftarrow w(l(N)) + w \cdot w(h(N))$. Set $m(h(N)) \leftarrow w$ (the earlier value of $m(h(N))$ was 1 because of the UpdateMessage procedure). We say that $h(N)$ has a *new message.*

     Remark: This is because the entire interval represented at $h(N)$ should be updated by multiplying the current weight by w. However, rather than applying this to every node in the subtree, we leave a message at $h(N)$ and only perform the updates when required in order to achieve the required time complexity.
     (b) At node N, if $l(N) \notin P_l$, then set $w(N) \leftarrow w(l(N)) + \cdot w(h(N))$.
  5. Symmetrically, at node ℓ_h, update $w(\ell_h) \leftarrow w(\ell_h) \cdot w$. For every internal node on the path P_h, *i.e.,* all nodes except for ℓ_h and $r_{l,h}$, the following updates are made in bottom-up order (going from ℓ_h to $r_{l,h}$).
     (a) At node N, if $l(N) \in P_h$, then set $w(N) \leftarrow w(l(N)) + \cdot w(h(N))$.
     (b) At node N, if $l(N) \notin P_l$, then set $w(N) \leftarrow w \cdot w(l(N)) + \cdot w(h(N))$. Set $m(l(N)) \leftarrow w$ (the earlier value of $m(l(N))$ was 1 because of the UpdateMessage procedure). We say that $l(N)$ has a *new message.*

6. Finally, along every node N on the path from $r_{l,h}$ to the root, set $w(N) \leftarrow w(l(N)) + w(h(N))$. (Here $r_{l,h}$ and the root are also updated.)

- Draw: The Draw procedure starts by picking x uniformly at random in $[0, 1)$ and moves from the root toward the leaves in order to find the leaf interval corresponding to x with respect to the distribution encoded by the tree $\mathcal{T}$. More precisely, the procedure starts from the root r and applies the UpdateMessage procedure at r, and then at $l(r)$ and $h(r)$. Then, define $\hat{x} = x \cdot w(r)$. The procedure moves to the subtree rooted at $l(r)$ if $\hat{x} < w(l(r))$ or to $h(r)$ otherwise and then proceeds recursively. At a given node N, the procedure applies UpdateMessage at both $l(N)$ and $h(N)$. It then moves toward $l(N)$ if $\hat{x} < w(l(N))$ or toward $h(n)$ if $\hat{x} > w(h(N))$. The procedure stops when it reaches a leaf ℓ and returns a point of $I(\ell)$ uniformly at random. See Figure 3.

We use the data structure in Algorithm 1 to represent the distributions p_t (we don't need to explicitly maintain the functions F_t, but if we did we could use a similar data structure). More precisely, the algorithm starts with a tree $\mathcal{T}$ containing a single node N_0 such that $I(N_0) = [0, 1)$ and $w(N_0) = 1$. Then at time t, the following are performed

1. In Step 2, x_t is drawn according to the Draw procedure described above.
2. In Step 4, the algorithm updates the tree after receiving f_t as follows: for interval $I_{t,j} = [a_{t,j-1}, a_{t,j})$ on which f_t is constant and takes value $v_{t,j}$, the procedure Update$(I_{t,j}, \exp(\eta v_{t,j}))$ is called.

The following claim results from the classical analysis of the complexity of interval trees (see (Cormen et al., 2009) for the complete proof).

Claim 3.4. *The running time of* UpdateMessage *is* $O(1)$. *Thus, by the definition of interval trees, the amortized running time of the* Insert, Draw *and* Update *procedures is* $O(\log(n))$ *for any tree of size* n.

Let $w_t(N)$ and $m_t(N)$ denote the values of $w(N)$ and $m(N)$ at the beginning of the t^{th} iteration. For any node N, define

$$\omega_t(N) = w_t(N) \cdot m_t(N) \cdot \prod_{N' \text{ ancestor of } N} m_t(N').$$

Lemma 3.5. *At the beginning of time step* t *and for any node* N,

$$\omega_t(N) = \sum_{\ell \text{ leaf descendant of } N} \omega_t(\ell). \tag{6}$$

Proof. We show this by induction on the level i of N in the tree. The Lemma holds for any leaf ℓ at level $i = 0$. Suppose it holds up to level $i - 1$ and consider a node at level i. We show, by induction on t, that for any node N of level i, $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$. If this is true, then by induction hypothesis Equation 6 holds. This is true for $t = 1$ as the only node is a leaf. We assume that this holds up to $t - 1$ and show that it holds for time t. Note that any call to UpdateMessage is made through a call to the Draw, Insert, or Update procedures, so we only consider that a call to one of those three procedures occurred at time t. Observe that in those three procedures, the UpdateMessage procedure is applied top-down from the root toward one (in the case of the Draw procedure) or two (in the case of the Insert and Update procedures) leaves. We consider a node N at distance at least one of all the paths followed by UpdateMessage and such that none of its ancestors received a new message at

time t. We show that the lemma holds for N. First if N is at distance at least 3 from the paths, we have that $\prod_{N' \text{ ancestor of } N} m_{t-1}(N') = \prod_{N' \text{ ancestor of } N} m_t(N')$ and $\omega_{t-1}(N) = \omega_t(N)$ since the messages are applied top-down and by definition of the UpdateMessage procedure and none of its ancestors received a new message. This holds for any descendant N' of N and so $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$ by induction hypothesis.

Now suppose that N is at distance 2 from the paths. We have that

$$m_{t-1}(N) \cdot \prod_{N' \text{ ancestor of } N} m_{t-1}(N') = m_t(p(N)) \cdot m_t(N).$$

Additionally, $w_t(N) = w_{t-1}(N)$. Thus, $\omega_t(N) = \omega_{t-1}(N)$. Since, by the above discussion, this also holds for $l(N)$ and $h(N)$. We have by induction hypothesis $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$.

We now consider a node N at distance exactly 1 from the paths. Since the UpdateMessage are applied top-down we have and none of its ancestor received a new message, $w_t(N) \cdot m_t(N) = w_{t-1}(N) \cdot m_{t-1}(N) \cdot \prod_{N' \text{ ancestor of } N} m_{t-1}(N') = \omega_{t-1}(N)$. We observe that $\prod_{N' \text{ ancestor of } N} m_t(N') = 1$. Thus, $\omega_t(N) = \omega_{t-1}(N)$. Since $l(N)$ and $h(N)$ are at distance 2, $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$ by induction hypothesis.

We now turn to prove that $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$ for the remaining nodes, *i.e.,* for the nodes in the paths and for the nodes for which an ancestor received a new message. Suppose first the Draw procedure was called at time t. We consider the path of nodes P taken by the procedure. For any node in the path we have $w_t(N) = w_{t-1}(N) \cdot m_{t-1}(N) \cdot \prod_{N' \text{ ancestor of } N} m_{t-1}(N') = \omega_{t-1}(N)$. Remark that $\prod_{N' \text{ ancestor of } N} m_t(N') = m_t(N) = 1$. Thus, $\omega_t(N) = \omega_{t-1}(N)$. Now, by the above discussion, the $\omega_t(l(N)) = \omega_{t-1}(l(N))$ and $\omega_t(h(N)) = \omega_{t-1}(h(N))$. Thus, $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$

Since the procedure also applies UpdateMessage to $l(N)$ and $h(N)$, we have $m(l(N)) = m(h(N)) = 1$. Thus, $\omega_t(l(N)) = w(l(N))$ and $\omega_t(h(N)) = w_t(h(N))$. Observe that the procedure sets $w_t(N) \leftarrow w_t(h(N)) + w_t(l(N))$ and therefore, $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$.

We now assume that a call to the Insert procedure occurred at time t. Since N is in the path we have that $\prod_{N' \text{ ancestor of } N} m_t(N') = m_t(N) = 1$ and the procedure sets $w_t(N) \leftarrow w_t(l(N)) + w_t(h(N))$. Since the procedure also applies UpdateMessage to N, we have $m(l(N)) = m(h(N)) = 1$. Thus, $\omega_t(l(N)) = w(l(N))$ and $\omega_t(h(N)) = w_t(h(N))$. Observe that the procedure sets $w_t(N) \leftarrow w_t(h(N)) + w_t(l(N))$ and therefore, $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$.

Therefore, we turn to the case where an update on an interval I_0 occurred at time t, *i.e.,* a call to function Update for an interval $I = [l, w]$ and a value w. Observe that for any node N such that $I(N) \cap I = \emptyset$, N is not on the paths and none of the ancestors of N received a new message. Thus, consider a node N such that $I(N) \cap I \neq \emptyset$. Suppose N is in P_l or in P_h or in the path between the root and $r_{l,h}$. Since UpdateMessage in applied top-down to all the ancestors of N and to N we have $\prod_{N' \text{ ancestor of } N} m(N') = m(N) = m(l(N)) = m(h(N)) = 1$, and so, $\omega_t(N) = w_t(N)$. Moreover, by definition of the procedure $w_t(N) = w_t(l(N)) + w_t(h(N)) = \omega_t(l(N)) + \omega_t(h(N))$ and thus, Equation 6 holds by induction hypothesis on the level of N.

Now suppose N is not in P_l or in P_h or in the path between the root and $r_{l,h}$. Consider the lowest ancestor N' of N in P_l or P_h and suppose, w.l.o.g, that $N' \in P_l$. Then since $N \notin P_l$ we have that $l(N') \in P_l$ and $h(N')$ is either N or an ancestor of N. In both cases we have that $\omega_t(N) = \omega_{t-1}(N) \cdot w$. Now, if N has a descendant N'', we have that $\omega_t(N'') = \omega_{t-1}(N'') \cdot w$ as well. Thus $\omega_t(N) = \omega_t(l(N)) + \omega_t(h(N))$. □

Lemma 3.6. *The Draw procedure produces a point p according to the distribution described at Step 1 of Algorithm 1.*

Proof. We start by showing the following invariant:

At any timestep t, for any leaf ℓ, for any $p \in I(\ell)$,

$$\omega_t(\ell) = |I(\ell)| \exp(\eta F_t(p)). \tag{7}$$

We proceed by induction on t. Note that for $t = 1$, it holds by definition. Suppose now that this holds up to time $t-1$ and consider time t. At time $t-1$ a call to the Update procedure was made, as otherwise by induction hypothesis, since no new message was inserted the invariant holds. Let $I = [l, h]$ be the interval that was updated and $\exp(\eta f_t(p))$ be the value of the update for some $p \in I$. Observe that, by the definition of the Update procedure, if a leaf ℓ is such that $I(\ell) \cap I = \emptyset$, then $\omega_t(N) = \omega_{t-1}(N)$.

Thus, consider a leaf ℓ such that $I(\ell) \cap I \neq \emptyset$. Suppose first $l \in I(\ell)$ or $h \in I(\ell)$. Then the procedure sets $w(\ell) \leftarrow w(\ell) \cdot \exp(\eta f_t(p))$ and since $\prod_{N' \text{ ancestor of } N} m(N') = m(N) = m(l(N)) = m(h(N)) = 1$, $w_t(\ell) = \omega_t(\ell)$. By Induction hypothesis, this means that $\omega_t(\ell) = |I(\ell)| \exp(\eta F_{t-1}(p)) \cdot \exp(\eta f_t(p)) = |I(\ell)| \exp(\eta(F_{t-1}(p) + f_t(p)))$. Since $F_t = F_{t-1} + f_t$, $\omega_t(\ell) = |I(\ell)| \exp(\eta(F_t(p))$.

Now consider a leaf ℓ such that $l \notin I(\ell)$ and $h \notin I(\ell)$ and $I(\ell) \cap I \neq \emptyset$. It follows that ℓ has an ancestor in either P_l or P_h. Consider the lowest ancestor N of ℓ in P_l or P_h and suppose, w.l.o.g, that $N \in P_l$. Observe $\prod_{N' \text{ ancestor of } N} m(N') = m(N) = m(l(N)) = m(h(N)) = 1$. Then since $\ell \notin P_l$ we have that $l(N) \in P_l$ and $h(N)$ is either ℓ or an ancestor of ℓ. In both cases we have that $\omega_t(\ell) = \omega_{t-1}(\ell) \cdot \exp(\eta f_t(p))$. Again, since $F_t = F_{t-1} + f_t$, $\omega_t(\ell) = |I(\ell)| \exp(\eta(F_t(p))$. We conclude that the invariant holds.

Now, observe that $F_t(x)$ is constant on any interval I such that there exists a leaf ℓ with $I(\ell) = I$. Denote by p_ℓ an arbitrary point of $I(\ell)$. Hence, combining Equations 6 and 7, for any node N,

$$\omega_t(N) = \sum_{\substack{\ell \text{ leaf of the subtree} \\ \text{rooted at } N}} |I(\ell)| \exp(\eta F_t(p_\ell)) = \int_a^b \exp(\eta F_t(x)) dx,$$

where $I(N) = [a, b]$. Moreover, remark that when the Draw procedure reaches a node N, for each ancestor N', we have $m(N') = 1 = m(N)$. It follows that we have $\omega_t(N) = w_t(N)$. Therefore, by Lemma 3.5, the procedure draws a point p according to the distribution described at Step 1 of Algorithm 1. □

From the previous Lemma and Claim 3.4, we deduce the following Theorem.

Theorem 3.7. *The* decision step *(step 2) of Algorithm 1 can be performed in time $O(\log Tk)$. Moreover the* update step *(step 4) can be performed in time $O(k \log(Tk))$.*

4 Bandit Setting

In the bandit setting, we only observe the value $f_t(x_t)$. Thus, we need to estimate f_t elsewhere. This is made difficult by the fact that f_t may have discontinuities. We construct an estimator $\hat{f}_t$ which takes an appropriately re-scaled value on a small interval around x_t and is 0 elsewhere. If this chosen interval is too large, then it is quite likely that a point of discontinuity of f_t lies in this interval. If it is small and no point of discontinuity of f_t lies in the interval, then $\hat{f}_t$ is an unbiased estimator of f_t; however, choosing too small an interval may make $\hat{f}_t(x_t)$ very large. Tuning the length of the interval and a careful analysis gives us a regret bound of $\tilde{O}(T^{2/3})$.

Theorem 4.1. *The expected regret of Algorithm 2 is bounded by (as long as $k\sigma\mu T = \omega(\log T)$),*

$$2\gamma T + 2\frac{\eta}{\mu}(e-2)T + \frac{1}{\eta}\ln\left(\mu^{-1}\right) + 2k\sigma\mu T + 1$$

Algorithm 2 Bandit Algorithm

input: $\eta,\ \mu,\ \gamma$ all positive and satisfying $1/\mu \in \mathbb{N}$, $\gamma \leq \frac{1}{2}$, $\eta \leq \gamma\mu$

set $\mathcal{I} = \langle [(i-1)\mu, i\mu) \rangle_{i=1}^{1/\mu}$ be a family of intervals.

set $w_1(x) = 1$ to be the constant 1 function over $[0,1)$

for $t = 1, 2, \ldots, T$ **do**

1. Define $p_t(x) = (1-\gamma)\frac{w_t(x)}{\int_0^1 w_t(x)dx} + \gamma$
2. Pick $x_t \sim p_t$
3. Let I_t be the interval of $\mathcal{I}$ that contains x_t.
4. Observe function $f_t(x_t)$. Receive payoff $f_t(x_t)$.
5. Set $\hat{f}_t(x) = \frac{f_t(x)}{p_t(I_t)}$ for all $x \in I_t$ and $\hat{f}_t(x) = 0$ for all $x \in [0,1) \setminus I_t$,
   Here for interval I, $p_t(I) = \Pr_{x\sim p_t}(x \in I)$.
6. Set $w_{t+1}(x) = w_t(x) \cdot \exp(\eta \hat{f}_t(x))$ for all $x \in [0,1)$.

The expectation is taken with respect to the random choices made by the algorithm as well as those by the adversary/nature. For suitable choices of μ, γ, and, η, this gives a regret bound of $O(\text{poly}(k, \sigma, \log(T))T^{2/3})$.

Proof. We follow the analysis as in the expert and semi-bandit cases. Let $W_t = \int_0^1 w_t$. We show that:

$$\ln\left(\frac{W_{t+1}}{W_t}\right) \leq \frac{\eta}{1-\gamma}\int_0^1 p_t(x)\hat{f}_t(x)dx + \frac{(e-2)\eta^2}{1-\gamma}\int_0^1 p_t(x)\hat{f}_t(x)^2 dx \tag{8}$$

By Definition of W_t,

$$\begin{aligned}
\frac{W_{t+1}}{W_t} &= \frac{\int_0^1 w_t(x)\exp(\eta\hat{f}_t(x))dx}{\int_0^1 w_t(x)dx} \\
&= \frac{1}{1-\gamma}\int_0^1 (p_t(x)-\gamma)\exp(\eta\hat{f}_t(x))dx &(9)\\
&\leq \frac{1}{1-\gamma}\int_0^1 (p_t(x)-\gamma)(1+\eta\hat{f}_t(x)+(e-2)\eta^2\hat{f}_t(x)^2)dx &(10)\\
&\leq 1 + \frac{\eta}{1-\gamma}\int_0^1 p_t(x)\hat{f}_t(x)dx + \frac{(e-2)\eta^2}{1-\gamma}\int_0^1 p_t(x)\hat{f}_t(x)^2 dx &(11)
\end{aligned}$$

Above (9) holds since $p_t(x) = (1-\gamma)\frac{W_t(x)}{\int_0^1 W_t(x)dx} + \gamma$, (10) holds as $\exp(z) \leq 1 + z + (e-2)z^2$ for $z \in [0,1]$—the requirement that $\eta \leq \gamma\mu$ implies that this is always satisfied, since $p(I_t) \geq \gamma\mu$ and $f_t(x_t) \leq 1$. Taking logarithms of both sides in (11) and using the fact that $\ln(1+z) \leq z$ we get (8).

Thus, by summing the inequality (8) for $t = 1, \ldots, T$, we get

$$\ln\left(\frac{W_{T+1}}{W_1}\right) \leq \frac{\eta}{1-\gamma}\sum_{t=1}^{T}\int_0^1 p_t(x)\hat{f}_t(x)dx + \frac{(e-2)\eta^2}{1-\gamma}\sum_{t=1}^{T}\int_0^1 p_t(x)\hat{f}_t(x)^2 dx \tag{12}$$

First, we observe that $\int_0^1 p_t(x)\hat{f}_t(x)dx = f_t(x_t)$. Also, we use the fact that

$$\int_0^1 p_t(x)(\hat{f}_t(x))^2 dx = f_t(x_t)\hat{f}_t(x_t) \leq \hat{f}_t(x_t)$$

Thus, we can rewrite (12) to give us:

$$\ln\left(\frac{W_{T+1}}{W_1}\right) \leq \frac{\eta}{1-\gamma}\sum_{t=1}^{T} f_t(x_t) + \frac{(e-2)\eta^2}{1-\gamma}\sum_{t=1}^{T}\hat{f}_t(x_t) \tag{13}$$

Note that since $w_1 \equiv 1$, $W_1 = 1$. Let x^* be some point in $[0,1)$ that is a maximizer of $\sum_{t=1}^{T} f_t(x)$. With probability 1, there exists x^* that lies in the interior of some interval defined by $\mathcal{I}$, denote this interval by $I(x^*)$. Since for all t, $\hat{f}_t$ is constant over $I(x^*)$, we have:

$$\ln\left(\frac{W_{T+1}}{W_1}\right) \geq \eta\sum_{t=1}^{T}\hat{f}_t(x^*) + \ln(|I(x^*)|) \tag{14}$$

Putting (13) and (14) together, rearranging some terms and dividing by η, we get:

$$\sum_{t=1}^{T}\hat{f}_t(x^*) - \sum_{t=1}^{T} f_t(x_t) \leq \frac{\gamma}{1-\gamma}\sum_{t=1}^{T} f_t(x_t) + \frac{(e-2)\eta}{1-\gamma}\sum_{t=1}^{T}\hat{f}_t(x_t) - \frac{\ln(|I(x^*)|)}{\eta} \tag{15}$$

We use the fact that $\gamma \leq 1/2$, thus $1/(1-\gamma) \leq 2$ to simplify some terms. By adding $\sum_{t=1}^{T} f_t(x^*)$ on both sides and rearranging terms, we get,

$$\sum_{t=1}^{T} f_t(x^*) - \sum_{t=1}^{T} f_t(x_t) \leq 2\gamma\sum_{t=1}^{T} f_t(x_t) + 2(e-2)\eta\sum_{t=1}^{T}\hat{f}_t(x_t) - \frac{\ln(|I(x^*)|)}{\eta} + \sum_{t=1}^{T} f_t(x^*) - \sum_{t=1}^{T}\hat{f}_t(x^*) \tag{16}$$

The LHS of (16) is exactly the regret suffered by the algorithm. Thus, to bound the expected regret, we only need to bound the expectation of the RHS, where the expectation is taken with respect to both the random choices of the algorithm and nature.

For the first term on the RHS of (16), we simply use the fact that $f_t(x_t) \leq 1$, hence the term is bounded by $2\gamma T$. For the second term, we first only look at the expectation with respect to the choice of $x_t \sim p_t$ to get,

$$\mathbb{E}_{x_t \sim p_t}[\hat{f}_t(x_t)] = \sum_{I \in \mathcal{I}} p_t(x_t \in I) \cdot \frac{f_t(x_t)}{p_t(x_t \in I)} \leq |\mathcal{I}| = \frac{1}{\mu}$$

and hence the expectation of the second term is bounded by $2(e-2)\eta T/\mu$. The intervals in $\mathcal{I}$ all have length μ. Thus, $-\ln(|I(x^*)|)/\eta = \ln(\mu^{-1})/\eta$.

Finally, we deal with the last term in the RHS of (16). However, there is a subtle issue which requires us to use the fact that we are playing against an *oblivious* adversary. The point x^* depends on the random choices made by nature/adversary (denoted by $\mathcal{F}_{adv}$). Let $\mathcal{H}_{t-1}$ denote the history of random choices made by the algorithm up to time $t-1$. Thus, we can compute $\mathbb{E}[\hat{f}_t(x^*) \mid \mathcal{F}_{adv}, \mathcal{H}_{t-1}]$, where the expectation is only with respect to the random choices made by the algorithm at time t. Let Z_t be 1 if the interval containing x^* contains a point of discontinuity of f_t and 0 otherwise. Note that conditioned on $\mathcal{F}_{adv}$, Z_t is not a random variable. However, we can conclude that,

$$f_t(x^*) - \mathbb{E}[\hat{f}_t(x^*) \mid \mathcal{F}_{adv}, \mathcal{H}_{t-1}] \leq Z_t,$$

where the expectation is only with respect to $x_t \sim p_t$. This holds since, if $Z_t = 0$, then $\mathbb{E}[\hat{f}_t(x^*) \mid \mathcal{F}_{adv}, \mathcal{H}_{t-1}] = f_t(x^*)$, else, the LHS is at most $f_t(x^*) \leq 1 = Z_t$.

Without conditioning on $\mathcal{F}_{adv}$, Z_ts are random variables, and we are interesting in bounding $\sum_{t=1}^{T} \mathbb{E}[Z_t]$. Rather than analyse the complex dependencies that arise because the interval containing x^* is itself a random variable that depends on the same random choices, we adopt a simpler approach. There are only $1/\mu$ possible intervals. Let $Z_{i,t}$ denote the random variable that the *ith* interval contains a point of discontinuity at time t. Note that due to the smoothed nature of the adversary this probability is at most $\mu\sigma k$, thus $\sum_{t=1}^{T} \mathbb{E}[Z_{i,t}] \leq \mu\sigma kT$. By Chernoff-bound the probability that $\sum_{t=1}^{T} Z_{i,t} \geq 3/2\mu\sigma kT$ is $\exp(-\Theta(\mu k\sigma T))$. Assuming $\mu k\sigma T = \omega(\log T)$ (which is the case for the choice considered), with probability at least $1/T$ by using a union bound for every interval $i = 1, \ldots, 1/\mu$, it is the case that $\sum_{t=1}^{T} Z_{i,t} \leq 2\mu k\sigma T$. In particular, this means that if Z_t correspond to the interval that contain x^*, $\sum_{t=1}^{T} \mathbb{E}[Z_t] \leq 2k\sigma\mu T + 1$. Thus, the expectation of the last term in the RHS of (16) can be bounded by $2k\sigma\mu T + 1$.

Putting everything together, we get,

$$\mathbb{E}[\text{Regret}(\text{Alg})] \leq 2\gamma T + 2\frac{\eta}{\mu}(e-2)T + \frac{1}{\eta}\ln\left(\mu^{-1}\right) + 2k\sigma\mu T + 1$$

Optimizing the RHS of the above equation with respect to μ, γ and η (while maintaining $\gamma \leq 1/2$ and $\eta \leq \gamma\mu$), gives that the expected regret is bounded by $O(\text{poly}(k, \sigma, \log(T))T^{2/3})$. □

Running Time. As in the full-information setting, the running time of our algorithm can be made polynomial in $\log(t)$ by using the *augmented* interval trees defined in Section 3.3.

5 Applications and Experiments

This section illustrates our framework with concrete examples. We describe three problems for which greedy approaches (also called *priority algorithms* by Borodin et al. (2003)) are often used in practice. We provide experimental results that highlight the efficiency of our approach.

We recall the definitions introduced by Gupta and Roughgarden (2016). An optimization problem Π is an *object assignment* problem if its input consists of a set of objects with so-called attributes and a solution to Π is a function $S : [n] \to \mathbb{R}_+$ where n is the number of objects, and $[n]$ denotes the set $\{1, \ldots, n\}$. For a given instance of an object assignment problem, the *attribute* of object i, ξ_i, is an ordered set of ℓ real values. Let $\Xi_1, \ldots, \Xi_\ell$ be the sets of possible values for attribute $1, \ldots, \ell$, respectively. For example, the knapsack problem is an object assignment problem: its input is a set of items with two attributes, value and size, in $\mathbb{R}_+$. A solution to the problem is a function $S : [n] \to \{0, 1\}$, where $S(i) = 1$ if and only if the ith item is part of the solution.

Define the functions $\lambda : \mathbb{R}_+ \times \Xi_1 \times \ldots \times \Xi_\ell \to \mathbb{R}_+$ as *single-parameter scoring rules*, where the first argument is the *parameter*. A *family* of single-parameter scoring rules is a set of scoring rules of the form $\lambda(\rho, \xi_i)$ for each parameter value ρ in some interval $I \subseteq \mathbb{R}$ and such that λ is continuous in ρ for each fixed value of ξ_i. We say that a family of single-parameter scoring rules is κ-*crossing* if for each $\xi_i, \xi_j, \xi_i \neq \xi_j$ there are at most κ values of ρ for which $\lambda(\rho, \xi_i) = \lambda(\rho, \xi_j)$.

Define an *assignment rule* α_λ as a function which given an object i and the value $\lambda(\rho, \xi_i)$ for some single-parameter scoring rule λ with parameter value ρ, computes the value of $S(i)$ and possibly modifies the attributes $\xi \setminus \{\xi_i\}$. We say that an assignment rule is β-*bounded* if for any object i, the number of different values that the attribute ξ can take is at most β.

We are now ready to introduce the notion of *greedy heuristic*. An algorithm A for an object assignment problem is a greedy heuristic if it greedily computes a solution S according to a

scoring-rule λ and an assignment rule α_λ. Thus, a family of κ-crossing scoring rules and a β-bounded assignment rule defines a $\kappa, \beta-$*family of greedy heuristics.*

Gupta and Roughgarden (2016) show that the outputs of a κ, β-family of greedy heuristics on an instance of size n is a piecewise constant function with $O((sn\beta)^2\kappa)$ pieces. Therefore we propose to apply our approach to the following problems.

5.1 Examples of Applications

We exhibit families of greedy heuristics for three classical NP-hard optimization problems.

The Knapsack problem: The input is a set of n pairs value and size, (v_i, s_i), and a capacity C. The objective is to output a subset $S \subseteq [n]$ such that $\sum_{j\in S} s_j \leq C$ and $\sum_{j\in S} v_j$ is maximum.

A family of greedy heuristics for Knapsack: Given a parameter ρ and an instance $\xi = \{\xi_1 = (v_1, s_1), \ldots, \xi_n = (v_n, s_n)\}$, the greedy heuristic performs the following computations. It first orders the elements by non-decreasing values of $v_i/(s_i)^\rho$. Then the heuristic greedily (subject to feasibility) adds objects to the solution in this order. Note that the cases $\rho = 0$ and $\rho = 1$ are classical heuristic for knapsack.

The Maximum Weighted Independent Set problem (MWIS): The input is a graph $G = (V, E)$, and weights w_i for each of the n vertices. The objective is to output a subset $S \subseteq [n]$ such that for any $i, j \in S$, $(i, j) \notin E$ and $\sum_{j\in S} w_j$ is maximum.

A family of greedy heuristics for MWIS: Denote by $N(i)$ the set of neighbors of i in G. Given a parameter ρ and an instance $\xi = \{\xi_1 = (w_1, N(1)), \ldots, \xi_n = (w_n, N(n))\}$, we consider the *adaptive* greedy heuristic. It adds all the degree 0 vertices to the solution and adds the vertex maximizing $w_i/|N(i)|^\rho$, then removes vertex i and all the vertices of $N(i)$ from the graph, updates $N(j)$ for the remaining vertices j, and repeats until there are no more vertices in the graph.

The Knapsack and MWIS problems were studied by Gupta and Roughgarden (2016). Here, we also consider the weighted k-means problem.

The weighted k-means problem : The input is a set of n points, a metric $d : [n] \times [n] \to \mathbb{R}_+$, and a set of weights $\{w_1, \ldots, w_n\}$. The objective is to output a subset $S \subseteq [n]$ of size k such that $\sum_{i\in[n]} \min_{j\in S} w_i \cdot d(i, j)^2$ is minimized.

A family of greedy heuristics for k-means: Given a parameter ρ and an instance $\xi = \{\xi_1 = (\{d(1, i) \mid i \in [n]\}), \ldots, \xi_n = (\{d(n, i) \mid i \in [n]\}))\}$, we consider the *adaptive* greedy heuristic, which can be seen as a generalization of Gonzalez' algorithm (Gonzalez, 1985) for the k-center problem (when $\rho = \infty$).

Algorithm 3 Adaptive Greedy for weighted k-means.

input: k, ρ, $\xi = \{\xi_1 = (\{d(1, i) \mid i \in [n]\}), \ldots, \xi_n = (\{d(n, i) \mid i \in [n]\})\}$,

set $S \leftarrow \emptyset$

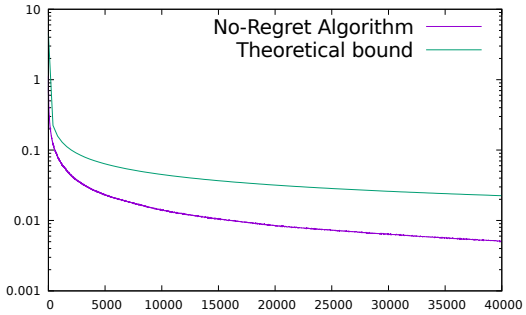
for $t = 1, 2, \ldots, k$ **do**

1. Add to S the point p that maximizes $\max_{i\in S} w_p/d(p, i)^{1/\rho}$.

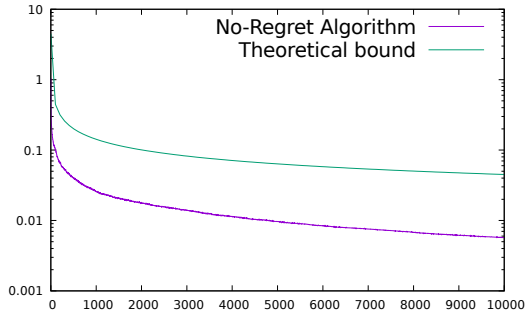
Return S

5.2 Experiments

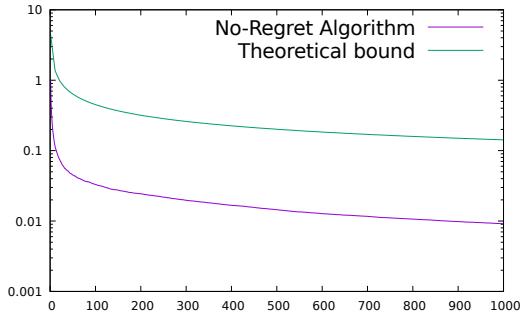
In this section, we describe experimental results to illustrate the efficiency of Algorithm 1.



(a) Knapsack



(b) MWIS



(c) Weighted k-means

Figure 4: Average per-round regret of Algorithm 1 for Knapsack, MWIS and weighted k-Means, and the theoretical bound given by Theorem 3.1. The x-axis represents the timesteps. The y-axis represents the per-round regret.

Knapsack: A random instance for knapsack consists of $n = 20$ real numbers for the values picked uniformly and independently at random in the interval $[0, 1)$ together with n real numbers for the sizes picked uniformly and independently at random in the interval $[0, 1]$ and a capacity bound of 1. We run Algorithm 1 for the family of greedy heuristics described in Section 5.1 for $\rho \in [0, 1]$ and for $T = 40000$ steps. We performed 100 rounds of the algorithm. Figure 4a shows the mean per-round regret of Algorithm 1 measured after t timesteps.

MWIS: For the MWIS problem, we consider random instances consisting of a random Erdős-Rényi graph with $n = 20$ vertices and with edge probability $p \in [0.1, 0.5]$ and n values for the weights picked uniformly and independently at random in the interval $[0, 1]$. We run the family of greedy heuristics described in section 5.1, with $\rho \in [0, 1]$ and for $T = 10000$. We performed 100 rounds of the algorithm. Again, Figure 4b shows the mean per-round regret of Algorithm 1 measured after t timesteps.

Weighted k-Means: In the experiments for weighted k-Means, we consider random instances of $n = 10$ points and where $k = 3$ defined as follows. The points lie in R^2 and are picked independently from k gaussians. The weights are picked uniformly and independently at random in the interval $[0, 1]$. We run the family of greedy heuristics described in section 5.1, with $\rho \in [0, 1]$ and for $T = 1000$ and again performed 100 rounds of the algorithm. Again, Figure 4c shows the mean per-round regret of Algorithm 1 measured after t timesteps.

Figures 4a,4b and 4c show that the per-round regret of Algorithm 1 decreases quickly and that the algorithm has significantly better performances on random instances than in the worst-case scenario. In all the cases, the variance introduced by the internal randomness of the No-Regret Algorithm is very small.

References

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003. ISSN 0097-5397. doi: 10.1137/S0097539701398375. URL http://dx.doi.org/10.1137/S0097539701398375.

Allan Borodin, Morten N. Nielsen, and Charles Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003. doi: 10.1007/s00453-003-1036-3. URL http://dx.doi.org/10.1007/s00453-003-1036-3.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory*, ALT'09, pages 23–37, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 3-642-04413-1, 978-3-642-04413-7. URL http://dl.acm.org/citation.cfm?id=1813231.1813240.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

Eugene Fink. How to solve it automatically: Selection among problem-solving methods. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 128–136. AAAI Press, 1998.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, pages 23–37, London, UK, UK, 1995. Springer-Verlag. ISBN 3-540-59119-2. URL http://dl.acm.org/citation.cfm?id=646943.712093.

Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi: 10.1016/0304-3975(85)90224-5. URL http://dx.doi.org/10.1016/0304-3975(85)90224-5.

Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 123–134, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4057-1. doi: 10.1145/2840728.2840766. URL http://doi.acm.org/10.1145/2840728.2840766.

Ling Huang, Jinzhu Jia, Bin Yu, Byung gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In J. Lafferty, C. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 883–891. 2010. URL http://books.nips.cc/papers/files/nips23/NIPS2010_1279.pdf.

Frank Hutter, Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods and evaluation (extended abstract). In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 4197–4201, 2015. URL http://ijcai.org/papers15/Abstracts/IJCAI15-595.html.

Satyen Kale, Chansoo Lee, and David Pal. Hardness of online sleeping combinatorial optimization problems. 2015.

Varun Kanade and Thomas Steinke. Learning hurdles for sleeping experts. *ACM Trans. Comput. Theory*, 6(3):11:1–11:16, July 2014. ISSN 1942-3454. doi: 10.1145/2505983. URL `http://doi.acm.org/10.1145/2505983`.

Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 681–690, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-047-0. doi: 10.1145/1374376.1374475. URL `http://doi.acm.org/10.1145/1374376.1374475`.

Robert D. Kleinberg. Nearly tight bounds for the continuum-armed bandit problem. In *18th Advances in Neural Information Processing Systems*, 2004.

Lars Kotthoff, Ian P. Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *AI Commun.*, 25(3):257–270, August 2012. ISSN 0921-7126. URL `http://dl.acm.org/citation.cfm?id=2350296.2350300`.

Gergely Neu and Michal Valko. Online combinatorial optimization with stochastic decision sets and adversarial losses. In *NIPS*, 2014.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

Jasper Snoek, Kevin Swersky, Richard Zemel, and Ryan P. Adams. Input warping for Bayesian optimization of non-stationary functions. In *31 st International Conference on Machine Learning*, Beijing, China, 2014.

Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, May 2004. ISSN 0004-5411. doi: 10.1145/990308.990310. URL `http://doi.acm.org/10.1145/990308.990310`.