SAPIENZA
UNIVERSITÀ DI ROMA

Online Caching Optimization with Predictions

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea Magistrale in Data Science

Candidate

Xufeng ZHANG
ID number 2015060

Thesis Advisors

Dr. Giovanni Neglia
Dr. Francescomaria Faticanti

Co-Advisor

Prof. Ioannis Chatzigiannakis

Academic Year 2022/2023

Thesis not yet defended

Online Caching Optimization with Predictions
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: zhangxufeng1998@gmail.com

Abstract

In computer science and information systems, online caching is a difficult problem that aims to get a optimal cache resources allocation to improve system performance. Traditional offline caching algorithms like LRU and LFU have a large theoretical bound for cost compared to best static policy when they are facing some online requests. Online learning algorithms have been introduced to address these issues with online caching. By operating sequentially, these algorithms enable adaptive decision-making based on prior observations and in-the-moment feedback, achieving a balance between exploiting long term historical observation and taking into account more recent request patterns. The FTPL (Follow the Perturbed Leader) algorithm is one of such methods.

Online learning algorithms can also take advantage of predictions about the future requests to improve their choices. The higher the accuracy, the larger the improvement. And when we talk about "optimistic" in online algorithms, which means we consider predictions. Such predictions could be generated by machine learning algorithms. With a higher accuracy, we would have better performance in online learning algorithms. When an online caching algorithm decides to change the set of items stored locally (the cache allocation), it needs to retrieve the missing items from a remote server. The update can then incur costs. These update costs are usually referred to as switching costs. This work examines the effects of predictions and switching cost while concentrating on the performance of the FTPL algorithm in the context of batched requests, which we process the requests after we accumulate a lot of them. First, we examine FTPL while accounting for switching costs but excluding predictions. We discover that the user utility loss increases linearly with the cost and the switching cost component does not increase with batch size. The results of batched algorithms show that algorithms which process with higher initial costs, their performance eventually tends to catch up to that of single requests. Then the performance of the OFTPL (Optimistic Follow the Perturbed Leader) algorithm without switching costs is examined. According to our analysis, the regret bound grows sublinearly with batch size. We also determine the ideal batch size with assumptions on requests and predictions.

After that, we take into account the OFTPL algorithm with switching costs. We find that it outperforms the FTPL algorithm with dynamic learning rates, also achieving a sublinear regret bound. But later we found that the algorithm with predictions does not necessarily perform better than the algorithm with higher switching costs

when the prediction accuracy is not high. Compared to the FTPL algorithm, perfect predictions lead to lower average costs, whereas predictions with lower accuracy result in rising average costs. These results highlight how critical high prediction accuracy is for the best performance in our scenario.

In conclusion, this work showed how the FTPL algorithm performs when batched requests, predictions, and switching costs are present. The findings provide direction for cache management strategies in online cases and aid in a better understanding of online caching algorithms.

Contents

1 Introduction **1**

1.1 Caching Problem 3

1.2 Related Works 5

1.3 Contribution 6

2 Online Learning Algorithms for Caching **8**

2.1 System Description 9

2.2 Online Gradient Descent 10

2.3 Online Mirror Descent 13

2.4 Follow the Leader 15

2.4.1 Follow the Regularized Leader 18

2.4.2 Follow the Perturbed Leader 21

2.5 Summary 27

3 Performance Analysis **29**

3.1 FTPL with Batched Requests and Switching Costs 29

3.2 OFTPL with Batched Requests 32

3.3 OFTPL with Switching Costs 35

3.4 OFTPL with Batched Requests and Switching Costs 44

4 Numerical Experiments **47**

4.1 Experiments Setting 47
4.2 Results 48
4.2.1 Comparison of different algorithms 48
4.2.2 Batched requests 49
4.2.3 OFTPL with switching cost 50
4.2.4 Experiment with CDN data 53
4.3 Summary 56

5 Conclusions **57**

Bibliography **59**

Chapter 1

Introduction

A cache is a high-speed storage layer storing frequently accessed data, improving data retrieval speed. Data enters the cache after being first fetched from primary storage, ensuring subsequent access is quicker. Caching is crucial for enhanced performance and reduced latency. Online caching is a challenging problem in computer science and information systems that aims to optimize the management and utilization of limited cache resources to improve system performance. Traditional offline caching algorithms assume full knowledge of the request sequence and are not directly applicable in online scenarios. The online caching problem involves deciding which items to store in a limited-size cache based on incoming requests. As requests arrive sequentially, the goal is to minimize the cost, which is generated from retrieving the file or switching costs, by making informed decisions on feedback and past, without prior knowledge of the future requests.

To address the caching problem, various traditional caching algorithms have been adapted, such as LRU, LFU. These algorithms make cache eviction decisions based on certain assumptions about item access patterns or cache occupancy. However, LFU lack the ability to adapt to changing workloads and access patterns, limiting their effectiveness in dynamic online environments[1]. And LRU can't learn enough patterns from the requests.

To overcome the challenges posed by the online caching problem, online learning algorithms have been introduced[2]. These algorithms provide a powerful framework for cache management by allowing adaptive decision-making based on past observations and real-time feedback. Online learning algorithms operate in a sequential manner, making decisions based on the observed history of requests and the feedback received. They aim to strike a balance between exploration and exploitation,

exploring new caching strategies while exploiting caching algorithms which maximize overall performance.

Online learning algorithms have emerged as effective tools for addressing the challenges of the online caching problem, enabling adaptive cache management strategies. In this work, we will present main online learning algorithms. And in particular, we will discuss the performance of FTPL (Follow the Perturbed Leader) in caching problem.

FTL (Follow the Leader) is a simple and intuitive online learning algorithm. It operates by selecting the caching strategy that has performed the best in the past, essentially following the leader[3]. FTL is easy to implement and computationally efficient, making it suitable for real-time applications. Its simplicity allows for fast adaptation, as it quickly adjusts to changing access patterns and workloads. However, FTL lacks exploration capabilities, because it always minimizes all historical information. It does not consider alternative caching strategies beyond the leader, potentially becoming stuck in suboptimal strategies if the leader performs poorly due to evolving access patterns. Additionally, FTL is sensitive to the initial conditions, as the initial choice of the leader can heavily influence its performance. If the initial leader performs poorly, it may take time for FTL to recover and adapt to better strategies.

To address the lack of exploration in FTL, FTRL (Follow the Regularized Leader) extends the algorithm by introducing regularization. It balances the exploitation of known effective caching strategies with the exploration of alternative strategies. FTRL assigns weights to different caching allocation and updates them based on observed performance, incorporating a regularization term to encourage exploration. The key benefit of FTRL is its ability to strike a balance between exploration and exploitation. By allowing for the discovery of new caching strategies while leveraging known effective strategies, FTRL offers improved performance in dynamic online environments[3]. Based on this algorithm, Mhaisen et al. [4] proved that it can achieve a sublinear regret ($O(\sqrt{T})$ in caching problem. However, FTRL still faces challenges. It requires tuning the regularization parameter, which can impact the algorithm's ability to adapt to different scenarios. Additionally, the computational complexity of FTRL is very high, which can limit its scalability in large-scale systems. And the projection operation consumes more cost than other methods. [5].

FTPL, on the other hand, introduces a perturbation-based approach to online

caching. It operates by randomly perturbing the caching decisions based on a random variable, thus exploring different caching strategies. The advantage of FTPL lies in its ability to achieve near-optimal performance in adversarial settings, making it robust against unknown and changing access patterns. FTPL is particularly well-suited for scenarios where the adversary can deliberately craft request sequences to challenge the caching algorithm. By perturbing the caching decisions, FTPL effectively explores different strategies and mitigates the impact of adversarial behavior. However, FTPL has some limitations. It may require additional computational resources due to the random perturbations, potentially affecting its scalability. Additionally, the randomness introduced by FTPL can make it more challenging to analyze and reason about its performance guarantees compared to deterministic algorithms like FTL and FTRL[6].

1.1 Caching Problem

In computer science, the difficulty of effectively managing cache memory to enhance data access and retrieval is referred to as the caching problem. To avoid accessing slower primary storage, like main memory or disk storage, caches are high-speed storage systems that store frequently accessed data or instructions. Caches, however, have a finite amount of space, so wise cache management choices are required. When space is limited, these decisions involve deciding which files to keep in the cache and which to remove. In the online caching problem, a cache hit occurs when a requested file is found in the cache, providing fast access. A cache miss happens when the requested file isn't in the cache, necessitating retrieval from a slower source. In order to improve system performance, the caching issue revolves around increasing cache hit rates and reducing cache misses.

Cache replacement policies are essential for cache management. These policies dictate which data items should be evicted from the cache when space is limited. Various cache replacement policies have been proposed, studied, and compared in the literature. The Least Recently Used (LRU) policy, for instance, evicts the item that has not been accessed for the longest time[7]. Other policies include Least Frequently Used (LFU), Random, and adaptive policies that dynamically adjust their behavior based on workload characteristics.

When we request a file from the cloud, and we have a correct prediction of the request, we can store the file in the cache in advance to get faster fetch speed. This is like moving files from storage to cache in a computer.

The difficulties of content delivery, caching, and network latency are greatly eased by the use of content distribution networks (CDN). A distributed network of cache spread over the network makes up CDN. The CDN minimises latency and network congestion by delivering content from the server that is closest to the user when they request it. By using caching techniques to store and serve content, CDN lighten the load on origin servers and boost the efficiency of content delivery[8].

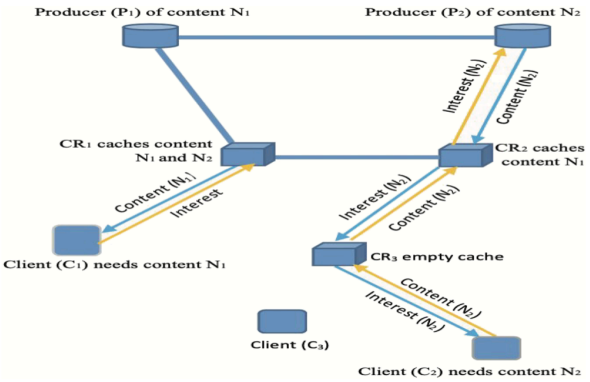

Figure 1.1. Example of content distribution network[9]

The way that CDNs work is by distributing content among edge servers that are carefully positioned all over the world for faster fetching of files. By ensuring that content is stored close to end users, this distribution minimises the distance that data must travel and increases delivery speeds. At the edge servers, CDNs use caching mechanisms to store frequently accessed content for quicker retrieval. The CDN determines whether content is cached before responding to a user request. In that case, the content is directly served from the cache, reducing latency and enhancing user experience. The CDN retrieves the content from the origin server and caches it for upcoming requests if it is not already present in one of the cache.

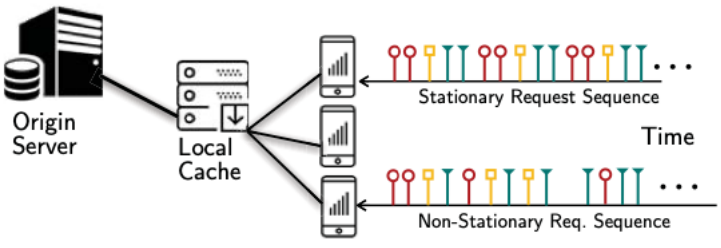

Figure 1.2. Example of caching problem[2]

1.2 Related Works

The problem of caching for content distribution is a complex one. Traditional caching policies cannot maintain good performance in the face of diverse requests. Paschos et al.[2] demonstrated that the Online Gradient Ascent policy is universally optimal because it ensures a regret, which is the maximum difference between the algorithm with the best static policy, that matches the lower bound. They also found it works well in bipartite network and showed that it has no regret, which means the average regret over the whole time horizon will converge to 0.

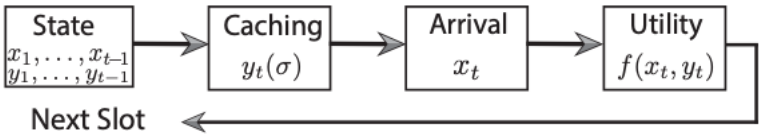

Figure 1.3. Caching decision process[2]

Si-Salem et al. [10] studied Online Mirror Descent (OMD)-based no-regret algorithms. They demonstrated how the request diversity in a batch affects the OMD strategy to be used and how OMD caching policies outperform conventional eviction-based policies.

Mhasien et al. [4] designed online caching algorithms for bipartite networks with fixed-size caches or elastic leased caches subject to time-average budget constraints based on the Follow-the-Regularized-Leader (FTRL) framework, which is further developed here to include predictions for the file requests. The forecasts are given by a content recommendation engine, which affects users' viewing behavior and, as a result, can naturally lessen the caching network's uncertainty about upcoming requests. They demonstrate that the suggested optimistic learning caching policies can achieve zero regret for perfect predictions and maintain the tightest possible bound even for arbitrary-bad predictions.

In their paper, they designed the first complete optimistic Follow-the-Perturbed leader policy that extends beyond the caching issue. They also look at the bipartite network caching problem and the possibility of caching files of various sizes. Finally, using extensive numerical experiments with traces from the real world, they assess the effectiveness of the suggested policies.

When we fetch the files form server to cache, the switch happens, and the corresponding cost is called switching cost. The switching cost also generates a new

part of regret compared to the setting without considering switching cost. And the switching cost contributes to the overall cost incurred by the system. The switching regret analysis of a Follow the Perturbed Leader-based anytime caching policy, which is demonstrated to have an order optimal switching regret, was introduced by Mukhopadhyay[11]. They found FTPL can get a sublinear regret even in the cases with switching costs when they choose a fixed learning rate. At the same time, they also proposed the possibility that it uses dynamic learning rate in algorithm. Finally they find it can still get a sublinear regret.

There also are some research on batched request. Faticanti et al. take into account various optimistic caching strategies that are based on the Follow-The-Regularized-Leader algorithm and have strong theoretical regret guarantees[12]. Because each update of the cache state necessitates the solution of a constrained optimization problem, these new policies have a higher computational cost than traditional ones like LRU and LFU. Then, in order to spread out the cost of the update over time or across several requests, they examine how well they perform when the cache is updated less frequently.

1.3 Contribution

Based on the above analysis, we are interested in the problem of batch processing and online caching algorithm with prediction, which is also the main contribution of this work. The main contributions are as follows:

1) For the FTPL algorithm that considers the swicthing cost and does not consider the predictions, this work finds that when the batch size changes, the regret of the switching cost part does not increase, and only a small part of the regret increases linearly with the increase of the batch size. We're still able to get sub-linear regrets.

2) When we use the OFTPL (Optimistic Follow the Perturbed Leader, FTPL with predictions) algorithm which does not consider the switching cost, the large batch size also makes the regret increase, following a square root relationship. Also we can get sub-linear regret in this case. At the same time, we consider adding some assumptions to the requests and predictions in section 3.2. In this case, we found that the batch size does not affect the regret bound in the optimal case. When predictions are perfect, we can get an optimal batch size. We also got the regret bound of OFTPL with switching cost in section 3.3, which is also a sublinear regret.

3) In the section 3.4, we consider both switching costs and predictions. We can see

that in this case we can still get sublinear regret and perform better than the same algorithm not considering predictions with a better regret bound. Finally, we also consider the possible effects of different switching costs and prediction accuracy.

Chapter 2

Online Learning Algorithms for Caching

Online learning algorithms are powerful tools in machine learning that enable decision-making in dynamic and evolving environments. Unlike traditional methods, where the entire dataset is available upfront, online learning algorithms operate sequentially, updating their models based on incoming data in a streaming fashion. They are particularly well-suited for scenarios where data arrives in a continuous stream and needs to be processed and learned in real-time.

The fundamental principle of online learning is to make predictions or take actions based on limited information at any given time. This is in contrast to offline learning, where the complete dataset is available for training and models are built to make predictions on new, unseen instances. Online learning algorithms, therefore, need to adapt and update their models incrementally as new data becomes available.

One of the key advantages of online learning algorithms is their ability to handle large-scale and streaming data efficiently. Rather than processing the entire dataset at once, online learning algorithms update their models iteratively, using a small batch or even individual data instances. This makes them highly scalable and suitable for real-time applications where data arrives at a high velocity.

Online learning algorithms typically follow a sequential update process. They start with an initial state and sequentially process data instances or mini-batches of data. For each instance, the algorithm makes a prediction or takes an action based on the current model, compares the prediction or action to the true value or feedback, and updates the model accordingly. This iterative process allows the algorithm to

adapt and improve its performance over time.

One of the key challenges in online learning is striking a balance between exploration and exploitation. As the algorithm updates its model based on incoming data, it needs to explore new patterns and update its beliefs while exploiting the information it has already learned. This trade-off is crucial for achieving optimal performance in online learning.

In the problem of online caching, the commonly used online algorithms are online gradient descent (OGD), online mirror descent (OMD), and follow the leader (FTL). They each have corresponding advantages [3]. Next I will introduce these algorithms.

2.1 System Description

We consider a caching system which is similar to system described in [11]. There is only one request at each time step. A file is requested by a single request and fetched from remote server. Upon a missing, the system occur a cost to retrieve the file. In the cases we consider a switching cost, if the cache state is updated, we would consider an additional switching cost. Assume there are N unique files in remote server. The capacity of local cache $C < N$.

Cache Configuration: The cache configuration at time t is represented by $\mathbf{y} \in \{1, 0\}^N$. It respect the capacity constraint:

$$\mathcal{Y} = \left\{ \boldsymbol{y} \in \{0, 1\}^N : \sum_{i \in [N]} y_i \leq C \right\} \tag{2.1}$$

Request: We consider an one-hot encoded request $\boldsymbol{x_t}$ at time t. Then we have $x_{t,i} = 1$, if file i is requested by one user. In batched requests, we still use the same notations, but we have $\sum_{i=1}^{N} x_{i,t} = m$ (batch size).

Cost: If file i is requested, $x_{t,i} = 1$, there is a exceptional retrieval cost:

$$f_{t,i}(\boldsymbol{x_t}) = w_i(1 - y_{t,i}) \tag{2.2}$$

$\boldsymbol{w}$ is the weight of cost for different files. At time t, file i is requested by user.

If we consider switching costs, we denote the cost to fetch a single file $D \geq 0$. Then

if r files are fetched into the cache, it will cost rD. Hence, the total switching cost between two time periods with cache state $\boldsymbol{y_t}, \boldsymbol{y_{t-1}}$, is $\frac{D}{2}\|\boldsymbol{y_t} - \boldsymbol{y_{t-1}}\|_1$.

Regret: As a rule in online learning, our goal is to design an algorithm that minimizes *regret*[13]. More precisely our goal is to get a sub-linear regret. Regret represents the difference between the loss expressed by our algorithm and by the optimal static method. Specifically, regret in our problem with switching cost is given by the following formula:

$$\mathbb{E}\left(R_T\right) = \sup_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}_t \rangle - \sum_{t=1}^{T} \mathbb{E}\left\langle \boldsymbol{y}_t, \boldsymbol{x}_t \right\rangle + \frac{D}{2} \sum_{t=1}^{T} \mathbb{E}\left\| \boldsymbol{y}_t - \boldsymbol{y}_{t-1} \right\|_1 \tag{2.3}$$

We want to use regret to get a consistently good result. When the algorithm has a sublinear regret, it means that its average regret will converge to 0 as the time horizon T becomes large.

Prediction: In this work, we do not have a strict definition of the prediction, it just needs to follow a similar form to $\boldsymbol{x_t}$. Many studies have confirmed that introducing predictions into online problems can improve the performance of algorithms[14]. And when the prediction accuracy is high enough, even 0 regret can be achieved[4]. In the case that predictions are inaccurate, algorithms are still able to maintain sublinear bounds. We called the algorithm is *optimistic* if it consider predictions.

Table 2.1. Notation

Notation	Defination
C	Cache capacity
N	Number of total files
$\mathbf{y_t}$	Cache configuration at time t
$\mathcal{Y}$	Set of cache configuration $\mathcal{Y} = \left\{ \boldsymbol{y} \in \{0,1\}^N : \sum_{i\in[N]} y_i \leq C \right\}$
$\mathbf{x_t}$	Request at time t
X_t	Cumulative file request up to time t $X_t = \sum_{\tau=1}^{t} \mathbf{x}_\tau$
γ	Gaussian random variables $\sim N(\mathbf{0}, \boldsymbol{I})$
η	Learning rate
m	Batch size
T	Time horizon

2.2 Online Gradient Descent

Consider a cost function $f_t(\mathbf{y_t})$, and let ∇f_t donate its gradient at $\mathbf{y_t}$.

Definition 2.1: The Online Gradient Descent (OGD) caching policy adjusts the decisions descending in the direction of the gradient:

$$\mathbf{y}_{t+1} = \Pi_y \left(\mathbf{y}_t - \eta_t \nabla f_t \right) \tag{2.4}$$

where η_t is the learning rate, and $\Pi_{\mathcal{Y}}(\mathbf{z}) = \arg\min_{\mathbf{y}\in\mathcal{Y}} \|\mathbf{z} - \mathbf{y}\|$ is the Euclidean projection of the argument vector $\mathbf{z}$ onto $\mathcal{Y}$, and $\|\cdot\|$ the Euclidean norm.

Next, look at the projection used in OGD. Follow [2], the projection can be written as a constrained quadratic program in online caching problem:

$$\begin{aligned} \Pi_{\mathcal{Y}}(z) \triangleq \arg\min_{y\geq 0} & \sum_{n=1}^{N} (z^n - y^n)^2 \\ \text{s.t.} & \sum_{n=1}^{N} y^n \leq C \text{ and } y^n \leq 1 \end{aligned} \tag{2.5}$$

where C is the cache size, N is file number.

By introducing Lagrangian, we have:

$$\begin{aligned} L(y, \rho, \mu, \kappa) = & \sum_{n=1}^{N} (z^n - y^n)^2 + \rho \left(\sum_{n=1}^{N} y^n - C \right) \\ & + \sum_{n=1}^{N} \mu_n \left(y^n - 1 \right) - \sum_{n=1}^{N} \kappa_n y_n, \end{aligned} \tag{2.6}$$

where ρ, μ, κ are the Lagrangian multipliers. The KKT condition of Equation (2.5) can be written as following sets:

$$\begin{aligned} &\mathcal{M}_1 = \{n \in \mathcal{N} : y^n = 1\}, \quad \mathcal{M}_2 = \{n \in \mathcal{N} : y^n = z^n - \rho/2\} \\ &\mathcal{M}_3 = \{n \in \mathcal{N} : y^n = 0\}, \end{aligned}$$

where $\rho = 2\left(|\mathcal{M}_1| - C + \sum_{n\in\mathcal{M}_2} z^n\right) / |\mathcal{M}_2|$.

For this optimization problem, Wang et al. [15] proposed an algorithm with time complexity $O(N^2)$. Paschos et al. adapted the algorithm for OGA algorithms that only requires $O(NlogN)$ steps ([2], Algorithm 1).

Next let's take a look at OGD's performance.

Theorem 2.1: For a fixed learning rate $\eta_t = \frac{\text{diam}(\mathcal{Y})}{L\sqrt{T}}$, the regret of OGD satisfies:

$$R_T(\text{OGA}) \leq \text{diam}(\mathcal{Y})L\sqrt{T} \tag{2.7}$$

Proof. Let's follow the steps from [2]. Firstly use the property of Euclidean projection, we have the bound as follow:

$$\begin{aligned}\left\|\boldsymbol{y}^{t+1} - \boldsymbol{y}^*\right\|^2 &\triangleq \left\|\Pi_{\mathcal{Y}}\left(\boldsymbol{y}_t + \eta_t \nabla f_t\right) - \boldsymbol{y}^*\right\|^2 \leq \left\|\boldsymbol{y}_t + \eta_t \nabla f_t - \boldsymbol{y}^*\right\|^2 \\ &= \left\|\boldsymbol{y}_t - \boldsymbol{y}^*\right\|^2 + 2\eta_t \nabla f_t^T\left(\boldsymbol{y}_t - \boldsymbol{y}^*\right) + \eta_t^2 \left\|\nabla f_t\right\|^2\end{aligned}$$

where $\boldsymbol{y}^*$ is the best static policy.

By fixing the learning rate η and summing up over T, we have:

$$\left\|\boldsymbol{y}_T - \boldsymbol{y}^*\right\|^2 \leq \left\|\boldsymbol{y}_1 - \boldsymbol{y}^*\right\|^2 + 2\eta \sum_{t=1}^{T} \nabla f_t^T\left(\boldsymbol{y}_t - \boldsymbol{y}^*\right) + \eta^2 \sum_{t=1}^{T} \left\|\nabla f_t\right\|^2$$

Because $\|\boldsymbol{y}_1 - \boldsymbol{y}^*\| \leq \text{diam}(\mathcal{Y})$ and $\|\nabla f_t\| \leq L$, we can get:

$$\sum_{t=1}^{T} \nabla f_t^T\left(\boldsymbol{y}^* - \boldsymbol{y}_t\right) \leq \frac{\text{diam}(\mathcal{Y})^2}{2\eta} + \frac{\eta T L^2}{2}$$

For a convex f_t, it has $f_t\left(\boldsymbol{y}_t\right) \geq f_t(y) + \nabla f_t^T\left(\boldsymbol{y}_t - \boldsymbol{y}^*\right)$. Finally we can use it to get the final bound:

$$\begin{aligned}R_T(OGD) &= \sum_{t=1}^{T}\left(f_t\left(\boldsymbol{y}^*\right) - f_t\left(\boldsymbol{y}_t\right)\right) = \sum_{t=1}^{T} \nabla f_t^T\left(\boldsymbol{y}_t - \boldsymbol{y}^*\right) \\ &\leq \frac{\text{diam}(\mathcal{Y})^2}{2\eta} + \frac{\eta T L^2}{2},\end{aligned}$$

If we choose a $\eta = \frac{\text{diam}(\mathcal{Y})}{L\sqrt{T}}$, we can get the sublinear bound as Theorem 2.1.

□

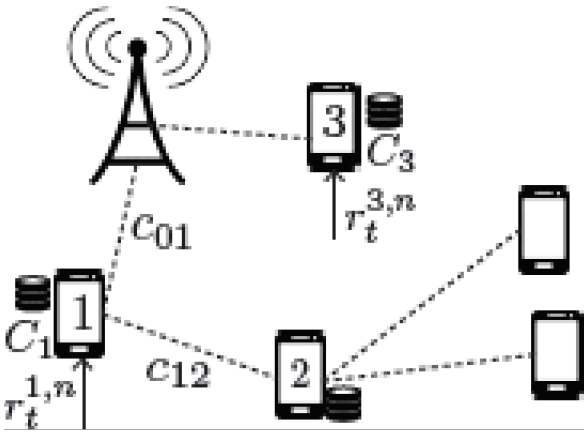

Figure 2.1. A BS-assisted D2D network.[16]

In addition to the single cache problem, OGD also has a good performance on some network cache problems. Paschos et al. successfully realized a sublinear regret in device-to-device cooperative network by using OGD algorithm without prior information[16].

2.3 Online Mirror Descent

Online mirror descent (OMD) is a class of scalable online learning techniques that use mirror maps to exploit data geometric structures[17]. Firstly, let's define a a differentiable mirror map Ψ:$\mathcal{D} \rightarrow \mathbb{R}$. It is strictly convex over $\mathcal{D}$ and ρ-strongly convex w.r.t. a norm $\|\cdot\|$ over $\mathcal{X} \cap \mathcal{D}$, where $\mathcal{X}$ is included in the closure of $\mathcal{D}$. And we have $\nabla\Psi(\mathcal{D}) = \mathbb{R}^N$.

Give a learning rate η and mirror map Ψ, the OMD steps can be described as follow:

1) Map the current state to dual space.

$$\hat{\mathbf{x}}_t = \nabla\Psi\left(\mathbf{x}_t\right) \tag{2.8}$$

2) Perform a gradient descent in the dual space.

$$\hat{\mathbf{y}}_{t+1} = \hat{\mathbf{x}}_t - \eta\nabla f_{\mathbf{r}_t}\left(\mathbf{x}_t\right) \tag{2.9}$$

3) Map $\hat{\mathbf{y}}_{t+1}$ to primal space.

$$\mathbf{y}_{t+1} = \left(\nabla\Phi\right)^{-1}\left(\hat{\mathbf{y}}_{t+1}\right) \tag{2.10}$$

4) Make a projection using the Bregman divergence associated with the mirror map Ψ.

$$\mathbf{x}_{t+1} = \prod_{\mathcal{X}\cap\mathcal{D}}^{\Psi} (\mathbf{y}_{t+1}) \tag{2.11}$$

where $\Pi^{\Psi}_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y}) = \arg\min_{\mathbf{x}\in\mathcal{X}\cap\mathcal{D}} D_{\Psi}(\mathbf{x},\mathbf{y})$ is the Bregman projection, $D_{\Psi}(\mathbf{x},\mathbf{y}) = \Psi(\mathbf{x}) - \Psi(\mathbf{y}) - \nabla\Psi(\mathbf{y})^T(\mathbf{x}-\mathbf{y})$ is the Bregman divergence.

In fact, we can get different algorithms and performances for different mirror maps. OMD is same to OGD when the mirror map is the Euclidean map[10].

If we choose $\Psi(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2$ and $\mathcal{D} = \mathbb{R}^N$, we have: $\nabla\Phi(\mathbf{x}) = \mathbf{x}$, for all $x \in \mathcal{D}$. Bergman distance is Euclidean distance in this case, $D_{\Phi}(\mathbf{x},\mathbf{y}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2$. Update step will be given by:

$$\mathbf{x}_{t+1} = \Pi_{\mathcal{X}}\left(\mathbf{x}_t - \eta\nabla f_{\mathbf{r}_t}(\mathbf{x}_t)\right), \forall t \in \{1,\ldots,T-1\} \tag{2.12}$$

where $\Pi_{\mathcal{X}}(\cdot)$ is the Euclidean projection . We can find that this is same to OGD we discussed before.

Theorem 2.2[10] For $\eta = \sqrt{\frac{k\left(1-\frac{k}{N}\right)}{\|\mathbf{w}\|_\infty^2\|\mathbf{r}\|_2^2 T}}$ the regret of *OGD*, satisfies:

$$\text{Regret}_T(\text{OGD}) \leq \|\mathbf{w}\|_\infty\|\mathbf{r}\|_2\sqrt{k\left(1-\frac{k}{N}\right)T} \tag{2.13}$$

Proof. The general bound of OMD is[18]:

$$\text{Regret}_T\left(\text{OMD}_{\Phi}\right) \leq \frac{D_{\Psi}\left(\mathbf{x}_*,\mathbf{x}_1\right)}{\eta} + \frac{\eta}{2\rho}\sum_{t=1}^{T}\|\nabla f_{\mathbf{r}_t}(\mathbf{x}_t)\|_*^2 \tag{2.14}$$

where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$, and $\mathbf{x}_*$ is the best static policy. The dual norm of Euclidean map is also the Euclidean one.

Let $\mathbf{x}_1$ be the minimizer of $\Psi(\mathbf{x})$, we have: $\nabla\Phi^T(\mathbf{x}_1)(\mathbf{x}-\mathbf{x}_1) \geq 0, \forall\mathbf{x} \in \mathcal{X}$ and $D_{\Psi}\left(\mathbf{x}_*,\mathbf{x}_1\right) \leq \Phi\left(\mathbf{x}_*\right) - \Psi\left(\mathbf{x}_1\right)$.

The minimum value of $\Psi(\mathbf{x})$ over $\mathcal{X}$ will be achieved when $x_i = \frac{k}{N}, i \in \mathcal{N}$, and $\mathbf{x}_*$ is

an integral solution. Then $\Psi(\mathbf{x}_*) = \frac{1}{2}k$ and we have:

$$D_\Phi(\mathbf{x}_*, \mathbf{x}_1) \leq \frac{1}{2}k(1 - k/N) \tag{2.15}$$

By bounding the secound part in Equation (2.14), we have:

$$\max_{r \in \mathcal{R}} \|\nabla f_{\mathbf{r}}(\mathbf{x})\|_2 \leq \max_{\mathbf{r} \in \mathcal{R}} \|\mathbf{w}\|_\infty \|\mathbf{r}\|_2 \tag{2.16}$$

By plugging the Equation (2.15), Equation (2.16) into Equation (2.14), and choose $\eta = \sqrt{\frac{k\left(1-\frac{k}{N}\right)}{\|\mathbf{w}\|_\infty^2 \|\mathbf{r}\|_2^2 T}}$ the regret of OGD, we will get the final bound in Equation (2.13). □

And OMD also can be same to FTRL (Follow the Regularized Leader) with specific mirror map, and we will discuss later.

2.4 Follow the Leader

Follow the Leader (FTL) tries to make decisions sequentially based on results that are observed in real-time. It is extensively used in disciplines including operations research, control theory, and machine learning[19].

The FTL algorithm operates iteratively, making decisions one at a time while taking into account the results of earlier judgements that have been observed. The fundamental concept is to "follow the leader" by making the decision that has so far produced the best results. By doing this, FTL makes use of the knowledge obtained from prior experiences and favours choices that have a history of success.

Follow the Leader is defined by:

Definition 2.2: Follow the Leader:

$$\forall t, \quad \mathbf{x}_t = \operatorname*{argmin}_{\mathbf{x} \in \mathcal{X}} \sum_{i=1}^{t-1} f_i(\mathbf{x}) \tag{2.17}$$

Through this definition, we can see that the performance of this method will be largely affected by the cost function f_t. Next, we analyze its possible performance

in some cases. But let's look at the Lemma 2.1 firstly.

Lemma 2.1: Let $x_1, \ldots, x_t$ be sequences produced by FTL then[3]:

$$R_T(\boldsymbol{x}) \leq \sum_{t=1}^{T} \left(f_t(x_t) - f_t(x_{t+1})\right) \tag{2.18}$$

Proof. If Equation (2.18) hold, by using the definition of Regret and assume u is a sequence in $\mathcal{X}$. Then we have:

$$\mathrm{R}_T(\mathbf{FTL}) = \sum_{t=1}^{T} \left(f_t(\mathbf{x}_t) - f_t(\mathbf{u})\right) \leq \sum_{t=1}^{T} \left(f_t(\mathbf{x}_t) - f_t(\mathbf{x}_{t+1})\right) \tag{2.19}$$

Rearranging the Equation (2.19), we can get:

$$\sum_{t=1}^{T} f_t(\mathbf{x}_{t+1}) \leq \sum_{t=1}^{T} f_t(\mathbf{u}) \tag{2.20}$$

If Equation (2.20) hold, we would get Lemma 2.1. Let's prove this equation.

Firstly, we assume Equation (2.20) hold for $T-1$.

$$\sum_{t=1}^{T-1} f_t(\mathbf{x}_{t+1}) \leq \sum_{t=1}^{T-1} f_t(\mathbf{u})$$

Adding $f_T(\mathbf{x}_{T+1})$ to both sides:

$$\sum_{t=1}^{T} f_t(\mathbf{x}_{t+1}) \leq f_T(\mathbf{x}_{T+1}) + \sum_{t=1}^{T-1} f_t(\mathbf{u})$$

We just assume u is a sequence in $\mathcal{X}$, so we can take $\mathbf{u} = \mathbf{x}_{T+1}$. Finally we have:

$$\sum_{t=1}^{T} f_t(\mathbf{w}_{t+1}) \leq \sum_{t=1}^{T} f_t(\mathbf{w}_{T+1}) = \min_{\mathbf{u} \in S} \sum_{t=1}^{T} f_t(\mathbf{u}) \tag{2.21}$$

From Equation (2.21) we can get Lemma 2.1. □

Theorem 2.3: Let $\mathcal{X} = \mathbb{R}^n$. Let $f_t(x) = \frac{1}{2}\|x - z_t\|^2$ for some $z_t \in \mathbb{R}^n$. Then in this case, FTL has $R(x_{1:T}) = O(\log T)$[3, Corollary 2.2]

Proof. x_t has closed form solution, $x_t = \frac{1}{t-1}\sum_{i=1}^{t-1} x_i$.

And we can rewrite:

$$\mathbf{x}_{t+1} = \frac{1}{t}(\mathbf{z}_t + (t-1)\mathbf{x}_t) = \left(1 - \frac{1}{t}\right)\mathbf{x}_t + \frac{1}{t}\mathbf{z}_t \tag{2.22}$$

We can get Equation (2.23) forEquation (2.22):

$$\mathbf{x}_{t+1} - \mathbf{z}_t = \left(1 - \frac{1}{t}\right)(\mathbf{x}_t - \mathbf{z}_t) \tag{2.23}$$

Then:

$$\begin{aligned} f_t(\mathbf{w}_t) - f_t(\mathbf{w}_{t+1}) &= \frac{1}{2}\|\mathbf{w}_t - \mathbf{z}_t\|^2 - \frac{1}{2}\|\mathbf{w}_{t+1} - \mathbf{z}_t\|^2 \\ &= \frac{1}{2}\left(1 - \left(1 - \frac{1}{t}\right)^2\right)\|\mathbf{w}_t - \mathbf{z}_t\|^2 \\ &\leq \frac{1}{t}\|\mathbf{w}_t - \mathbf{z}_t\|^2 . \end{aligned}$$

Let $L = \max_t \|z_t\|$, since w_t is average of Z_i, we have:

$$\sum_{t=1}^{T}(f_t(\mathbf{w}_t) - f_t(\mathbf{w}_{t+1})) \leq (2L)^2 \sum_{t=1}^{T}\frac{1}{t}$$

Using Lemma2.1 and inequality $\sum_{t=1}^{T} 1/t \leq \log(T) + 1$, we will get this theorem.

□

FTL fails when you are dealing with the online linear optimization as shown in [3], Example 2.3. And if there is much noise in the past, It's hard to get a good result by FTL. We know that FTL is frequently a "greedy" algorithm that takes advantage of the best choices that may be inferred from previous results. While this may result in strong performance in the short term, it may also prevent the algorithm from exploring novel and maybe superior alternatives.

To avoid above problem, there are some popular variations include algorithms like FTRL (Follow the Regularized Leader), FTPL (Follow the Perturbed Leader) and

Online Mirror Descent. These variations aim to improve robustness, adaptivity, exploration-exploitation balance, and provide stronger theoretical guarantees. And the variations of FTL are what we will discuss next.

2.4.1 Follow the Regularized Leader

A well-known online optimisation approach called Follow the Regularised Leader (FTRL) expands the idea of Follow the Leader (FTL) by using regularisation methods. It is frequently used in machine learning applications, especially for online learning jobs where judgements must be made instantly and the data is sent sequentially.

By including a regularisation term to the decision-making process, FTRL overcomes the FTL algorithm's drawbacks of being noise-sensitive and lacking an exploration-exploitation balance. By placing restrictions or penalties on the decision distribution, this regularisation term promotes diversity and lowers the chance of overfitting to noisy or ambiguous data. Regularisation improves FTRL's ability to manage noisy or non-stationary situations and adapt to changing circumstances.

Firstly, the definition of update step of FTRL is:

Definition 2.3: Follow the Regularized Leader:

$$\forall t, \quad \mathbf{x}_t = \underset{\mathbf{x}\in S}{\operatorname{argmin}} \sum_{i=1}^{t-1} f_i(\mathbf{x}) + R(\mathbf{x}) \tag{2.24}$$

Definition 2.4: Let's consider a problem as following:

$$\boldsymbol{x}_t \in \underset{\boldsymbol{x}\in S}{\operatorname{argmin}} R(x) - \langle \boldsymbol{x}, \theta_t \rangle, \quad \theta_t = -\eta \sum_{i=1}^{t-1} \ell_i, \quad \ell_i \in \partial f_i(x_i)$$

when we choose $\tilde{\boldsymbol{x}}_{t+1}$ so that $\nabla R(\tilde{\boldsymbol{x}}_{t+1}) = \nabla R(\tilde{\boldsymbol{x}}_t) - \eta \ell_t$, if we don't use $\tilde{\boldsymbol{x}}_t$ in practice, it is called *Lazy OMD*.

As we said before, OMD has some connections with FTRL, and in some cases their predictions are equivalent. Let's look at this case.

Lemma 2.2: Let cost funtion is linear, Lazy OMD and FTRL produce identical predictions[20]:

$$\arg\min_{\mathbf{x}\in\mathcal{K}} \{B_R(\mathbf{x}\|\mathbf{y}_t)\} = \arg\min_{\mathbf{x}\in\mathcal{K}} \left(\eta \sum_{s=1}^{t-1} \nabla_s^\top \mathbf{x} + R(\mathbf{x})\right) \tag{2.25}$$

where B_R is Bregman divergence.

Proof. From unconstrained minimum:

$$\mathbf{x}_t^\star \overset{\text{def}}{=} \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\{\sum_{s=1}^{t-1} \nabla_s^\top \mathbf{x} + \frac{1}{\eta} R(\mathbf{x})\right\} \tag{2.26}$$

It satisfies:

$$\nabla R(\mathbf{x}_t^\star) = -\eta \sum_{s=1}^{t-1} \nabla_s \tag{2.27}$$

By definition we know $\mathbf{y}_t$ also satisfies Equation (2.27) and $R(x)$ is strictly convex. So there is only one solution: $\mathbf{y}_t = \mathbf{x}_t^\star$.

Then,

$$\begin{aligned} B_R(\mathbf{x}\|\mathbf{y}_t) &= R(\mathbf{x}) - R(\mathbf{y}_t) - (\nabla R(\mathbf{y}_t))^\top (\mathbf{x} - \mathbf{y}_t) \\ &= R(\mathbf{x}) - R(\mathbf{y}_t) + \eta \sum_{s=1}^{t-1} \nabla_s^\top (\mathbf{x} - \mathbf{y}_t) \end{aligned} \tag{2.28}$$

Since $R(\mathbf{y}_t)$ and $\sum_{s=1}^{t-1} \nabla_s^\top \mathbf{y}_t$ are independent of $\mathbf{x}$, we would know that $B_R(\mathbf{x}\|\mathbf{y}_t)$ is minimized at the point $\mathbf{x}$ that minimizes $R(\mathbf{x}) + \eta \sum_{s=1}^{t-1} \nabla_s^\top \mathbf{x}$ over $\mathcal{K}$. Finally we will have:

$$\arg\min_{\mathbf{x}\in\mathcal{K}} B_R(\mathbf{x}\|\mathbf{y}_t) = \arg\min_{\mathbf{x}\in\mathcal{K}} \left\{\sum_{s=1}^{t-1} \nabla_s^\top \mathbf{x} + \frac{1}{\eta} R(\mathbf{x})\right\} \tag{2.29}$$

□

In actual problem, we need to choose different kind of regularizer ($R(\mathbf{x})$). Let's start from the definition of regularizers. We use following definition from [21]:

$$r_0(\boldsymbol{x}) = \boldsymbol{I}_{\mathcal{X}}(\boldsymbol{x}), \quad r_t(\boldsymbol{x}) = \frac{\sigma_t}{2} \|\boldsymbol{x} - \boldsymbol{x}_t\|^2, t \geq 1 \tag{2.30}$$

where $\|\cdot\|$ is L2 norm, and $\boldsymbol{I}_{\mathcal{X}}(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{X}$ and ∞ otherwise. And we use following parameters if we consider the prediction in time t ($\widetilde{c}_t$):

$$\sigma_t = \sigma\left(\sqrt{h_{1:t}} - \sqrt{h_{1:t-1}}\right), \sigma_1 = \sigma\sqrt{\boldsymbol{h}_1}, \boldsymbol{h}_t = \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2 \tag{2.31}$$

where $\sigma \geq 0, c_t = \nabla f_t(x_t)$ and $h_{1:t} = \sum_{i=1}^{t} h_i$. The update step of OFTRL (we introduce the predictions and consider it is optimistic FTRL) is:

$$\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x}\in\mathbf{R}^m} \left\{ r_{0:t}(\boldsymbol{x}) - \left(c_{1:t} + \tilde{\boldsymbol{c}}_{t+1}\right)^\top x \right\} \tag{2.32}$$

Theorem 2.4: The regret bound of OFTRL is[4]:

$$R_T \leq 2\sqrt{2(1+JC)}\sqrt{\sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2} \tag{2.33}$$

Proof. From ([22], Theorem 1), we have:

$$R_T \leq r_{1:T}(\boldsymbol{x}^\star) + \sum_{t=1}^{T} \|\boldsymbol{c}_t - \widetilde{\boldsymbol{c}}_t\|^2_{(t),\star}, \quad \forall x^\star \in \mathcal{X} \tag{2.34}$$

where $r_{0:t}(\boldsymbol{x})$ is 1strongly-convex w.r.t. some norm $\|\cdot\|_{(t)}$.

In our case, $r_{1:t}$ is 1-strongly-convex w.r.t. norm $\|\boldsymbol{x}\|_{(t)} = \sqrt{\sigma_{1:t}}\|\boldsymbol{x}\|$ which has dual norm $\|\boldsymbol{x}\|_{(t),\star} = \|\boldsymbol{x}\|/\sqrt{\sigma}_{1:t}$. Using the definition in Equation (2.31), we have $\sigma_{1:t} = \sigma\sqrt{h_{1:t}}$. Then from Equation (2.34) we can get:

$$R_T \leq \frac{\sigma}{2}\sqrt{h_{1:T}}D^2_{\mathcal{X}} + \sum_{t=1}^{T} \frac{\boldsymbol{h}_t}{\sigma\sqrt{h_{1:t}}} \tag{2.35}$$

For $\mathbf{x}$, we can bound the norm term as follow:

$$\begin{aligned} \|\boldsymbol{x} - \boldsymbol{x}_t\|^2 &= \sum_{n,j} \left(y_{nj} - y^t_{nj}\right)^2 + \sum_{n,i,j} \left(z_{nij} - z^t_{nij}\right)^2 \\ &\overset{(a)}{\leq} \sum_{n,j} \left|y_{nj} - y^t_{nj}\right| + \sum_{n,i,j} \left|z_{nij} - z^t_{nij}\right| \overset{(b)}{\leq} 2(JC+1) \triangleq D^2_{\mathcal{X}} \end{aligned} \tag{2.36}$$

Finally, we can use $\sigma = 2/D_{\mathcal{X}}$ and ([23],Lemma 3.5) to get the result. □

FTRL (Follow the Regularized Leader) is an algorithm that minimizes a cumulative loss function by iteratively making decisions based on observed outcomes. It incorporates regularization to control decision complexity and prevent overfitting. We know that FTRL overcome some shortage of FTL. However, it has limitations in terms of complexity, computational overhead. Next, I will introduce a simpler algorithm, which will also be the main algorithm used later.

2.4.2 Follow the Perturbed Leader

Follow the Perturbed Leader (FTPL) expands on the idea of Follow the Leader (FTL) by adding randomization or perturbations to the decision-making process. It is frequently employed in situations where decision-making must adjust to ambiguous or shifting environments.

The FTPL algorithm makes decisions iteratively based on results observed in order to reduce cumulative loss. The decision selection in FTPL, in contrast to FTL, includes perturbations or randomizations. By introducing noise into the decision-making process, it facilitates exploration and helps reduce excessive dependence on past information.

The update step of FTPL can be defined as:

Definition 2.5: Follow the Perturbed Leader:

$$\forall t, \quad \mathbf{x}_t = \underset{\mathbf{x}\in S}{\operatorname{argmin}} \sum_{i=1}^{t-1} f_i(\mathbf{x}) + \eta\gamma \tag{2.37}$$

where η is learning rate and γ is random noisy.

Now let's discuss performance in caching problems. We assume that the disturbance term $((\gamma))$ will initially be generated by a standard normal distribution. The sequence of predictions at each step will be generated as follows:

$$\boldsymbol{y}_t \leftarrow \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}_t + \eta_t\boldsymbol{\gamma}\rangle$$

where $\boldsymbol{X}$ is the accumulated requests. We consider y_t is cache state in time t in this part.

Theorem 2.5 The regret bound for FTPL in caching problem is[24]:

$$\mathbb{E}_{\{\gamma_t\}_{t\geq 1}}\left(R_T^{FTPL}\right) \leq 1.51(\log N)^{1/4}\sqrt{CT}$$

Proof. Define the potential function:

$$\Phi_\eta(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\gamma}\sim\mathcal{N}(0,I)}\left[\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{y}, \boldsymbol{x}+\eta\boldsymbol{\gamma}\rangle\right] \tag{2.38}$$

Using the Taylor's expansion of Φ, we have:

$$\Phi\left(X_{t+1}\right) = \Phi\left(X_t\right) + \left\langle \nabla\Phi\left(X_t\right), x_t\right\rangle + \frac{1}{2}\left\langle x_t, \nabla^2\Phi\left(\tilde{x}_t\right) x_t\right\rangle \tag{2.39}$$

for some $\tilde{x}_t$ on the line segment connecting X_t and $X_{t+1} = X_t + x_t$.

Thus,

$$\mathbb{E}\left[\langle y_t, x_t\rangle\right] = \Phi\left(X_{t+1}\right) - \Phi\left(X_t\right) - \frac{1}{2}\left\langle x_t, \nabla^2\Phi\left(\tilde{x}_t\right) x_t\right\rangle \tag{2.40}$$

Summing up over T, we have:

$$\mathbb{E}\left(\boldsymbol{R}_T\right) \leq \Phi_\eta\left(\boldsymbol{X}_1\right) + \frac{1}{2}\sum_{t=1}^{T}\left\langle \boldsymbol{x}_t, \nabla^2\Phi_\eta\left(\tilde{\boldsymbol{x}}_t\right)\boldsymbol{x}_t\right\rangle \tag{2.41}$$

First part can be bound as follow[25]:

$$\Phi_\eta\left(X_1\right) \leq \eta\sqrt{2C\log\binom{N}{C}} \leq C\eta\sqrt{2\log N}, \tag{2.42}$$

For second part, we have:

$$\left\langle \boldsymbol{x}_t, \nabla^2\Phi_\eta\left(\tilde{\boldsymbol{x}}_t\right)\boldsymbol{x}_t\right\rangle \leq \max_{i,\boldsymbol{x}}\left(\left|\nabla^2\Phi_\eta(\boldsymbol{x})\right|\right)_i \tag{2.43}$$

From Lemma 7 in [26]:

$$\left(\nabla^2\Phi_\eta(\boldsymbol{x})\right)_{ij} = \frac{1}{\eta}\mathbb{E}\left[\hat{y}\left(\tilde{\boldsymbol{x}}_t + \eta\boldsymbol{\gamma}\right)_i\gamma_j\right] \tag{2.44}$$

where $\hat{y}(z) \in \arg\max_{\boldsymbol{y}\in\dagger}\langle \boldsymbol{y}, z\rangle$. Finally, we get:

$$\mathbb{E}\left(R_T\right) \leq C\eta\sqrt{2\log N} + \frac{T}{\eta\sqrt{2\pi}} \tag{2.45}$$

If we choose $\eta = \frac{1}{(4\pi\log N)^{1/4}}\sqrt{\frac{T}{C}}$, we will have:

$$\mathbb{E}_{\{\gamma_t\}_t}\left(R_T^{\mathrm{FTPL}}\right) \leq 1.51(\log N)^{1/4}\sqrt{CT} \tag{2.46}$$

□

In the above problem, we did not consider the introduction of predictions like OFTRL. Similarly, let's analyze the performance of OFTPL.

Theorem 2.6: The regret of OFTPL is:

$$\mathbb{E}_\gamma\left[R_T\right] \leq 3.68\sqrt{C}\left(\ln\frac{Ne}{C}\right)^{1/4}\sqrt{\sum_{t=1}^{T}\|\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\|_1^2}. \tag{2.47}$$

where for any time horizon T and $N \geq 2C$ with $C \geq 11$[21, Theorem 3].

Proof. Similarly, we can use potential function to get Equation (2.48).

$$\Phi_t\left(X_t\right) = \Phi_t\left(X_{t-1} + \tilde{\boldsymbol{x}}_t\right) + \left\langle \nabla\Phi_t\left(X_{t-1} + \tilde{\boldsymbol{x}}_t\right), \boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\right\rangle + \frac{1}{2}\left\langle \boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t, \nabla^2\Phi_t\left(\hat{\boldsymbol{x}}_t\right)\left(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\right)\right\rangle \tag{2.48}$$

From the convexity of $\Phi_t(\cdot)$, we have:

$$\Phi_t\left(X_{t-1} + \tilde{\boldsymbol{x}}_t\right) \leq \Phi_t\left(X_{t-1}\right) + \left\langle \nabla\Phi_t\left(X_{t-1} + \tilde{\boldsymbol{x}}_t\right), \tilde{\boldsymbol{x}}_t\right\rangle \tag{2.49}$$

From Equation (2.48) and Equation (2.49) we can get:

$$\Phi_t\left(X_t\right) \leq \Phi_t\left(X_{t-1}\right) + \left\langle \nabla\Phi_t\left(X_{t-1} + \tilde{\boldsymbol{x}}_t\right), \boldsymbol{x}_t\right\rangle + \frac{1}{2}\left\langle \boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t, \nabla^2\Phi_t\left(\hat{\boldsymbol{x}}_t\right)\left(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\right)\right\rangle. \tag{2.50}$$

Then,

$$\Phi_T(X_T) \leq \sum_{t=1}^{T} \left(\Phi_t(X_{t-1}) - \Phi_{t-1}(X_{t-1}) + \mathop{\mathbb{E}}_{\gamma}[\langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle] + \frac{1}{2} \left\langle \boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t, \nabla^2 \Phi_t(\hat{\boldsymbol{x}}_t)(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) \right\rangle \right) \tag{2.51}$$

Summing up over T, the bound can be written as follow:

$$R_T \leq \Phi(X_T) - \sum_{t=1}^{T} \mathbb{R}[\langle \boldsymbol{x}_t, \boldsymbol{y}_t \rangle] \leq \sum_{t=1}^{T} \left(\Phi_t(X_{t-1}) - \Phi_{t-1}(X_{t-1}) + \frac{1}{2} \|\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\|_{H_t} \right). \tag{2.52}$$

First part can be bound as follow:

$$\begin{aligned} &\sum_{t=1}^{T} \Phi_t(X_{t-1}) - \Phi_{t-1}(X_{t-1}) = \sum_{t=1}^{T} \mathop{\mathbb{R}}_{\gamma}[\Phi(X_{t-1} + \eta_t \gamma) - \Phi(X_{t-1} + \eta_{t-1}\gamma)] \\ &\overset{(a)}{\leq} \sum_{t=1}^{T} \mathop{\mathbb{E}}_{\gamma}[\Phi((\eta_t - \eta_{t-1})\gamma)] \overset{(b)}{\leq} \sum_{t=1}^{T} (\eta_t - \eta_{t-1}) \mathop{\mathbb{E}}_{\gamma}[\Phi(\gamma)] \\ &\leq \eta_T \mathop{\mathbb{R}}_{\gamma}\left[\max_y \langle y, \gamma \rangle\right] \overset{(c)}{\leq} \eta_T \sqrt{2C \ln \binom{N}{C}} \overset{(d)}{\leq} \eta_T C \sqrt{2 \ln(Ne/C)}, \end{aligned} \tag{2.53}$$

where (a) and (b) follow from the sub-linearity of the potential function; (c) from Massart's lemma. (d) from $\binom{N}{C} \leq \left(\frac{Ne}{C}\right)^C$.

In order to bound the second part, we need to bound Hessian matrix item firstly.

$$\left|H_{i,j}^t\right| = \frac{1}{\eta_t} \left| \mathop{\mathbb{E}}_{\gamma}[\tilde{\boldsymbol{y}}(\hat{\boldsymbol{x}} + \eta_t \gamma)_i \gamma_j] \right| \leq \frac{1}{\eta_t} \mathop{\mathbb{E}}_{\gamma}[|\tilde{\boldsymbol{y}}(\hat{\boldsymbol{x}} + \eta_t \gamma)_i| |\gamma_j|] \leq \frac{1}{\eta_t} \mathop{\mathbb{E}}_{\gamma}[|\gamma_i|] \leq \frac{1}{\eta_t} \sqrt{\frac{2}{\pi}}, \tag{2.54}$$

Then second part can be bounded as:

$$\begin{aligned} &\|\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\|_{H_t} = \langle \boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t, H_t(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) \rangle = \sum_{i,j} (x_{t,i} - \tilde{x}_{t,i}) H_{ij}^t (x_{t,j} - \tilde{x}_{t,j}) \\ &\overset{(a)}{\leq} \sum_{i,j} \left| (x_{t,i} - \tilde{x}_{t,i}) \left\| H_{ij}^t \right\| (x_{t,j} - \tilde{x}_{t,j}) \right| \overset{(b)}{\leq} \frac{1}{\eta_t} \sqrt{\frac{2}{\pi}} \left(\sum_i |(x_{t,i} - \tilde{x}_{t,i})| \right)^2 = \frac{1}{\eta_t} \sqrt{\frac{2}{\pi}} \|\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\|_1^2 \end{aligned} \tag{2.55}$$

Actually there is another way to bound second part:

$$\begin{aligned}
&\frac{1}{2}\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_{H_t}=\Phi_t\left(X_t\right)-\Phi_t\left(X_{t-1}+\tilde{\boldsymbol{x}}_t\right)-\left\langle\nabla\Phi_t\left(X_{t-1}+\tilde{\boldsymbol{x}}_t\right),\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\rangle \\
&=\mathop{\mathbb{E}}_{\gamma}\left[\Phi\left(X_t+\eta_t\gamma\right)-\Phi\left(X_{t-1}+\tilde{\boldsymbol{x}}_t+\eta_t\gamma\right)\right]+\left\langle\nabla\Phi_t\left(X_{t-1}+\tilde{\boldsymbol{x}}_t\right),\tilde{\boldsymbol{x}}_t-\boldsymbol{x}_t\right\rangle\overset{(a)}{\leq}\Phi\left(\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right)+ \\
&+\left\langle\nabla\Phi_t\left(X_{t-1}+\tilde{\boldsymbol{x}}_t\right),\tilde{\boldsymbol{x}}_t-\boldsymbol{x}_t\right\rangle=\max_{\boldsymbol{y}\in\mathcal{X}}\left\langle\boldsymbol{y},\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\rangle+\left\langle\nabla\Phi_t\left(X_{t-1}+\tilde{\boldsymbol{x}}_t\right),\tilde{\boldsymbol{x}}_t-\boldsymbol{x}_t\right\rangle\overset{(b)}{\leq}2\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1
\end{aligned} \tag{2.56}$$

Based on these ways, we can bound second part:

$$\frac{1}{2}\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_{H_t}\leq\min\left(\frac{1}{\sqrt{2\pi}}\frac{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}{\eta_t},2\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1\right) \tag{2.57}$$

Then,

$$\begin{aligned}
\frac{1}{2}\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_{H_t}&\leq\min\left(\frac{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}{\sqrt{2\pi}\beta\sqrt{\sum_{\tau=1}^{t-1}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}},2\frac{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}{\sqrt{2\pi}\beta\sqrt{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}}\right) \\
&\overset{(a)}{\leq}\frac{3}{\sqrt{2\pi}\beta}\frac{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}{\sqrt{\sum_{\tau=1}^{t}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}}
\end{aligned} \tag{2.58}$$

where (a) is $\min\left(a_1/a_2,b_1/b_2\right)\leq\frac{a_1+a_2}{b_1+b_2}$ for two positive fractions. $\eta_t=\beta\sqrt{\sum_{\tau=1}^{t-1}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}$, $t\geq1$, and $0<\beta\leq\frac{1}{\sqrt{2\pi}}$.

The final bound we get is:

$$\mathop{\mathbb{B}}_{\gamma}\left[R_T\right]\leq\eta_T C\sqrt{2\log(Ne/C)}+\frac{3}{\sqrt{2\pi}\beta}\sum_{t=1}^{T}\frac{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}{\sqrt{\sum_{\mathrm{r}=1}^{t}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}}. \tag{2.59}$$

Since second part can be bounded as:

$$\begin{aligned}
\sum_{t=1}^{T}\frac{\left\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\right\|_1^2}{\sqrt{\sum_{\tau=1}^{t}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}}&\leq\sum_{t=1}^{T}\int_{\sum_{\tau=1}^{t-1}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}^{\sum_{\tau=1}^{t}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}\frac{dx}{\sqrt{x}}=\int_0^{\sum_{\tau=1}^{T}\left\|\boldsymbol{x}_\tau-\bar{\boldsymbol{x}}_\tau\right\|_1^2}\frac{dx}{\sqrt{x}} \\
&=2\sqrt{\sum_{\tau=1}^{T}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}
\end{aligned} \tag{2.60}$$

we can get the final result:

$$\mathbb{E}_\gamma\left[R_T\right]\leq\sqrt{\sum_{\tau=1}^{T}\left\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\right\|_1^2}(C\sqrt{2\ln(Ne/C)})\beta+\frac{6}{\beta\sqrt{2\pi}}\Bigg) \tag{2.61}$$

To get a minimum value, we can choose $\beta = \sqrt{\frac{3}{C}}\left(\frac{1}{\pi \ln(Ne/C)}\right)^{1/4}, C \geq 11.$ □

We can see that when this algorithm considers predictions, its performance depends on the accuracy of the predictions. When we have a perfect prediction, the theoretical regret boundary is 0.

Now we consider switching cost without taking into account predictions. We have noticed that in our previous questions, we have always assumed that switching between two different caching configurations has no cost. Now we consider that moving a unit of files will generate a cost of D. We can get its performance through the following theorem.

Theorem 2.7: With switching cost, the regret bound of FTPL is[11]:

$$\mathrm{E}\left(R_T\right) \leq \frac{2}{\pi^{1/4}}\sqrt{C(D+1)}(\ln(N/C))^{1/4}\sqrt{T}, \tag{2.62}$$

Proof. The regret bound can be written as:

$$\begin{aligned}\mathbb{E}\left(R_T\right) = \sup_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{y}, \boldsymbol{X}_t\rangle - \sum_{t=1}^{T}\mathbb{E}\left(\boldsymbol{y}_t, \boldsymbol{x}_t\rangle\right. \\ + \frac{D}{2}\sum_{t=1}^{T}\mathbb{E}\left\|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\right\|_1\end{aligned} \tag{2.63}$$

The first part and the second part of Equation (2.63) we have proved in Theorem 2.5. We only need to bound third part. And note that we are considering a discrete caching problem.

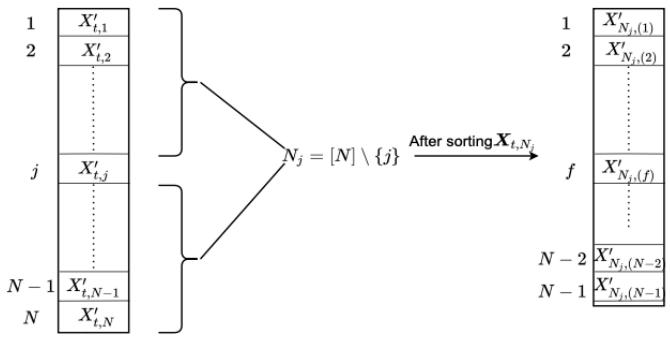

Figure 2.2. Switching cost

We have $\mathbb{E}_{\boldsymbol{\gamma}}\left[\|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\|_1\right] = 2\mathbb{P}\left(\boldsymbol{y}_t \neq \boldsymbol{y}_{t-1}\right)$, because either $\|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\|_1 = 0$ or $\|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\|_1 = 2$. For single cache problem, we consider the case in Figure 2.2, where $X'_{t,f} = X_{t,f} + \eta\gamma_f$, $X'_{N_j,(f)}$ denote the f_{th} component of the sorted vector when we ignore the file j. Then,

$$\begin{aligned}\mathbb{P}\left(\boldsymbol{y}_{t+1} \neq \boldsymbol{y}_t\right) =& \mathbb{P}\left(X'_{N_{f_t},(C)} \geq X'_{t,f_t} > X'_{N_{f_t},(C)} - 1\right) \\ =& \mathbb{P}\left(X_{N_{f_t},(C)} - X_{t,f_t}\right)/\eta \geq \gamma_{f_t} > \left(X'_{N_{f_t},(C)} - X_{t,f_t}\right)/\eta\right) - 1/\eta\Big) \\ =& \mathbb{E}\left(\mathbb{E}\mathbb{P}\left(X'_{N_{f_t},(C)} - X_{t,f_t}\right)/\eta \geq \gamma_{f_t} > \left(X'_{N_{f_t},(C)} - X_{t,f_t}\right)/\eta\right) - 1/\eta \mid X'_{N_{f_t},(C)}\right)\end{aligned} \tag{2.64}$$

By definition of γ, we get:

$$\mathbb{P}\left(\boldsymbol{y}_{t+1} \neq \boldsymbol{y}_t\right) \leq \frac{1}{\sqrt{2\pi}\eta} \tag{2.65}$$

From Theorem 2.5 and Equation (2.65), we have:

$$\begin{aligned}\mathbb{E}\left(R_T\right) &= C\eta\sqrt{2\ln(N/C)} + \frac{T}{\eta\sqrt{2\pi}} + \frac{DT}{\eta\sqrt{2\pi}} \\ &= C\eta\sqrt{2\ln(N/C)} + \frac{T}{\eta\sqrt{2\pi}}(1+D).\end{aligned} \tag{2.66}$$

Finally, we can choose $\eta = \sqrt{T(D+1)/C}(4\pi\ln(N/C))^{-1/4}$ to get the minimum regret bound. Then we could get the bound in Theorem 2.7. □

We can see that the algorithm of FTPL is simpler and easier to calculate than FTRL, and it can also overcome the shortcomings of FTL. So in this work, I will start from the FTPL algorithm and analyze the possible performance in different cases.

2.5 Summary

All these algorithms for online optimisation have their own advantages and disadvantages. OGD offers a straightforward and effective method, but it may be lacking in robustness and exploration abilities. By adding a mirror map, OMD expands OGD, allowing for greater flexibility and accommodating a wider variety of loss functions. FTL is simple, but noise-sensitive and has an unbalanced exploration-exploitation ratio. By incorporating regularisation techniques and striking a balance between exploitation and regularisation, FTRL addresses these limitations. In order to ex-

plore different options and adapt to uncertain or dynamic environments, FTPL introduces randomization.

In reality, the choice of algorithm is influenced by the peculiarities of the problem, the characteristics of the data, and the trade-offs between computational robustness, exploration, and efficiency. Based on the needs and limitations of the current optimisation task, researchers and practitioners frequently select one of these algorithms.

Table 2.2. Summary of different algorithms in different cases

Algorithm	Conditions	Guarantees		Complexity	Discrete/Continuous	Source
		Best case	Worst case			
OMD	Single cache	o($\sqrt{T}$)		o(N)	Continuous	[24]
FTPL-based uncoded caching policy	Single cache	o (poly-log(N)$\sqrt{T}$)		o(N)	Continuous	[24]
OGA	Single cache	o($\sqrt{T}$)		o(N2)	Continuous	[2]
FTPL	Bipartite network	o (poly-log(N)$\sqrt{T}$)		o(N)	Discrete	[27]
TBGRD	General network	o (poly-log(N)$\sqrt{T}$)		o(N)	Continuous	[28]
FTRL	Single cache, prediction	o($\sqrt{T}$)		o(N)	Continuous	[4]
FTRL	Bipartite network, prediction	o($\sqrt{T}$)		o(N)	Continuous	[4]
OFTRL	Single cache, prediction	0	o($\sqrt{T}$)	o(N2)	Discrete	[21]
OFTPL	Single cache, prediction	0	o (poly-log(N)$\sqrt{T}$)	o(N)	Discrete	[21]
OFTPL-unequal	Single cache, prediction, unequal size	0	o($\sqrt{T}$)	o(N2)	Discrete	[21]
OFTRL-unequal	Single cache, prediction, unequal size	0	o (poly-log(N)$\sqrt{T}$)	o(N)	Discrete	[21]
OFTRL	Bipartite network, prediction	0	o($\sqrt{T}$)	o(N2)	Discrete	[21]
Experts	Single cache, prediction	b<0	$R_T^{(w)}$ + o($\sqrt{T}$)	o(N)	Discrete	[21]

Chapter 3

Performance Analysis

The Follow the perturbed leader (FTPL), as shown in Algorithm 1, is an algorithm used in online learning and decision-making. It operates in adversarial environments, where an agent must repeatedly choose actions without knowing future outcomes. FTPL selects an action by perturbing the accumulated information, encouraging exploration and preventing over-reliance on a single expert. The algorithm has theoretical guarantees and achieves sublinear regret, making it effective in adapting to changing environments. FTPL strikes a balance between exploration and exploitation, making it valuable for decision-making in adversarial settings.

3.1 FTPL with Batched Requests and Switching Costs

Batched requests of Follow the Perturbed Leader refer to a modified version of the algorithm that updates the cache only after having collected a batch of requests. Instead of making decisions for each request independently, FTPL processes them in batches. It offers several advantages, such as improved computational efficiency and reduced communication overhead. By processing multiple requests together, it enables parallel processing, making it particularly beneficial in large-scale and high-frequency scenarios. But when we consider batched case, its theoretical performance would be different from single case. We can see the result got in Theorem 3.1.

Algorithm 1 The FTPL Caching Policy with Swithcing Cost[11]

Learning rate $\{\eta_t\}_{t\geq 1}$, switching cost $D \geq 0$, cache capacity C, initial cache-configuratiot $\boldsymbol{y}_0$
$\boldsymbol{X}_1 \leftarrow \mathbf{0}$
Sample: $\boldsymbol{\gamma} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
for t=1 *to* T **do**
    Cache the top C files corresponding to the perturbed cumulative count vector $\boldsymbol{X}_t + \eta_t \boldsymbol{\gamma}$, i.e.,
        $\boldsymbol{y}_t \leftarrow \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \langle \boldsymbol{y}, \boldsymbol{X}_t + \eta_t \boldsymbol{\gamma} \rangle$
    User requests a file corresponding to the request vector $\boldsymbol{x}_t$
    The policy receives a reward $q_t = \langle \boldsymbol{y}_t, \boldsymbol{x}_t \rangle - \frac{D}{2} \|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\|_1$
    Update $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{X}_t + \boldsymbol{x}_t$
end for

Theorem 3.1. The regret of FTPL with switching cost in the batched case is upper-bounded as:

$$E\left(R_T\right) \leq C\eta\sqrt{2\log N} + \frac{mT}{\eta\sqrt{2\pi}} + min\{\frac{CT}{m}, \frac{DT}{\eta\sqrt{2\pi}}\} \tag{3.1}$$

where m is the batch size, N is the number of files, D is the switching cost to fetch 1 file into cache, C is the cache size.

Proof. Define the potential function:

$$\Phi_\eta(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\gamma}\sim N(0,I)}\left[\max_{\boldsymbol{y}\in\boldsymbol{y}} \langle \boldsymbol{y}, \boldsymbol{x} + \eta\boldsymbol{\gamma} \rangle\right] \tag{3.2}$$

With noise variance η is upper bounded as Equation (2.41):

$$\mathbb{E}\left(R_{T/m}\right) \leq \Phi_\eta\left(\boldsymbol{X_1}\right) + \frac{1}{2}\sum_{i=1}^{T/m} \left\langle \boldsymbol{x}_i, \nabla^2\Phi_\eta\left(\tilde{\boldsymbol{x}}_i\right)\boldsymbol{x}_i \right\rangle, \tag{3.3}$$

The first term would be bounded as follow:

$$\Phi_\eta\left(\boldsymbol{X_1}\right) \leq \eta\sqrt{2C\log\binom{N}{C}} \leq C\eta\sqrt{2\log N} \tag{3.4}$$

The second term is:

$$\left\langle \boldsymbol{x}_i, \nabla^2 \Phi_\eta\left(\tilde{x}_i\right) \boldsymbol{x}_i \right\rangle \leq m^2 \times \max_{i,j,\boldsymbol{x}} \left(\left| \nabla^2 \Phi_\eta(\boldsymbol{x}) \right| \right)_{ij} \tag{3.5}$$

Using Jensen's inequality we have that ([24], Theorem 3):

$$\left(\left| \nabla^2 \Phi_\eta(\boldsymbol{x}) \right| \right)_{ij} \leq \frac{1}{\eta} \mathbb{E}\left[\left| \hat{y}\left(\tilde{\boldsymbol{x}}_i + \eta \boldsymbol{\gamma} \right)_i \right| |\gamma_j| \right] \overset{(a)}{\leq} \frac{1}{\eta} \mathbb{E}\left[|\gamma_j| \right] \overset{(b)}{=} \frac{1}{\eta} \sqrt{\frac{2}{\pi}}, \tag{3.6}$$

where (a) follows from the fact that for all $\boldsymbol{y} \in \mathcal{Y}$, we have $y_i \in \{0, 1\}$, and the (b) follows from the fact that $\gamma_i \sim N(0, 1)$. Now we have an up bound for regret without switching cost. $E(R_T) \leq C\eta\sqrt{2 \log N} + \frac{1}{2} \frac{T}{m} \frac{m^2}{\eta} \sqrt{\frac{2}{\pi}}$

The third part of the switching cost can be bound as follow:

We know that in single-request case, regret for switching cost is $\frac{D}{\eta\sqrt{2\pi}}$[11]. Let's define the set of all files excepting the j^{th} file by $N_j = [N] \backslash \{j\}, \forall j \in [N]$. For each moved file in descending order of X' ($X'_{t,f} = X_{t,f} + \eta\gamma_f$, f is the order of file, and $X_t = \sum_{i=1}^{t-1} x_i$). f_t is the number that has the largest X'_{t,f_t} which will be fetched at $t+1$, but not at t.

For further analysis, let's consider how the switching cost changes when we request twice at once. If both requests are the same and it is the file which will fetched in next time, we have:

$$\begin{aligned}
\frac{1}{2} \mathbb{E}_{\boldsymbol{\gamma}} \left[\| \boldsymbol{y}_t - \boldsymbol{y}_{t-1} \|_1 \right] &\leq \mathbb{P}\left(X'_{N_{f_t},(C)} \geq X'_{t,f_t} > X'_{N_{f_t},(C)} - 2 \right) \\
&= \mathbb{P}\left(X_{N_{f_t},(C)} - X_{t,f_t} \right) / \eta \geq \gamma_{f_t} > \left(X'_{N_{f_t},(C)} - X_{t,f_t} \right) / \eta - 2/\eta \\
&\leq \frac{1}{\sqrt{2\pi}} \int_{X'_{N_{f_t},(C)} - X_{t,f_t})/\eta - 2/\eta}^{(X_{N_{f_t},(C)} - X_{t,f_t})/\eta} 1 du = \frac{2}{\eta\sqrt{2\pi}}
\end{aligned}$$

Next we consider another case. There are 2 files that need to be moved from server to cache.

$$\frac{1}{2} \mathbb{E}_{\boldsymbol{\gamma}} \left[\| \boldsymbol{y}_t - \boldsymbol{y}_{t-1} \|_1 \right] \leq \sum_{i=1}^{2} P\left(X'_{N_{f_t+i-1},(C-2+i)} \geq X'_{t,f_t+i-1} > X'_{N_{f_t+i-1},(C-2+i)} - 1 \right) \leq \frac{2}{\eta\sqrt{2\pi}}$$

We can find that the possible cost of moving a newly requested file into the cache is the same as the cost of an additional request for a file that is about to enter the cache at the next time. Based on the above analysis, we can conclude that when we consider a batch of requests with size m, we can at most get the possible switching cost of $\frac{m}{\eta\sqrt{2\pi}}$ in one step, but it should not exceed C.

Finally, the regret in this case is upper-bounded as:

$$E\left(R_T\right) \leq C\eta\sqrt{2\log N} + \frac{mT}{\eta\sqrt{2\pi}} + min\left\{\frac{CT}{m}, \frac{DT}{\eta\sqrt{2\pi}}\right\} \tag{3.7}$$

□

We can see that in this case, we can still get a sublinear regret when we take $\eta \propto \sqrt{T}$. But if the batch size is too large, the regret might not be sublinear.

3.2 OFTPL with Batched Requests

Algorithm 2 The OFTPL Caching Policy without Switching Cost[21]

Input: $\eta_1 = 0, y_1 = \arg\min_{y\in X} \langle y, x_1\rangle$
Output: $\boldsymbol{y_t} \in \mathcal{Y}$
Sample: $\boldsymbol{\gamma} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
for t=2,3... *to* T **do**
    $\tilde{\boldsymbol{x}}_{\boldsymbol{t}} \leftarrow$ prediction
    $\eta_t = \frac{1.3}{\sqrt{C}}\left(\frac{1}{\ln(Ne/C)}\right)^{\frac{1}{4}}\sqrt{\sum_{\tau=1}^{t-1}\|x_\tau - \tilde{x}_\tau\|_1^2}$
    $\boldsymbol{y_t} = \text{argmax}_{y\in X}\langle \boldsymbol{y}, X_{t-1} + \tilde{\boldsymbol{x}}_{\boldsymbol{t}} + \eta_t\gamma\rangle$
    The policy receives a reward $q_t = \langle \boldsymbol{y}_t, \boldsymbol{x}_t\rangle$
    $X_t = X_{t-1} + \boldsymbol{x_t}$
end for

When we introduce predictions, the optimistic FTPL algorithm (OFTPL) can get a better bound when we have good predictions[21]. We consider the OFTPL algorithm in Algorithm 2. Note that there is no switching cost in this part. Similarly, let us analyze the performance of the above algorithm under batched requests.

Theorem 3.2. The regret of OFTPL without switching cost in batched cases is upper-bounded as:

$$\mathbb{E}[R_T] \leq 2\sqrt{mT}\left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right) \tag{3.8}$$

Proof. From ([21], Theorem 3), we have the regret in unbatched case.

$$\mathbb{E}_\gamma\left[R_T\right] \leq \sqrt{\sum_{\tau=1}^{T} \|\boldsymbol{x_\tau} - \tilde{\boldsymbol{x}}_{\boldsymbol{\tau}}\|_1^2} \left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right) \tag{3.9}$$

In batched requests,

$$\begin{aligned}
\mathbb{E}_\gamma\left[R_T\right] &\leq \sqrt{\sum_{\tau=1}^{T/m} \|\boldsymbol{x_\tau} - \tilde{\boldsymbol{x}}_{\boldsymbol{\tau}}\|_1^2} \left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right) \\
&= \sqrt{\sum_{\tau=1}^{T/m} \left(\sum_{i=1}^{N} |x_{\tau,i} - \tilde{x}_{\tau,i}|\right)^2} \left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right). \\
&\leq \sqrt{\sum_{\tau=1}^{T/m} (2m)^2} \left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right) \\
&= 2\sqrt{mT} \left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right)
\end{aligned}$$

□

When we do not make any assumptions about requests and predictions, we can get the upper bound above. We can see that as the batch size increases, our bounds also increase. Next, we make assumptions about the distribution of the request and prediction sequences, then we can get the results of Lemma 3.1.

Lemma 3.1. Assume $x_{\tau,i} \sim Poisson(\lambda_i m)$, m_0 is a first warm-up interval. we have 2 conclusions in OFTPL:

- with perfect predictor, batch size does not affect our regret.
- consider predictor as a random variable, we have the optimal batch size $m_0/2$.

Proof. Look at the Equation (3.9) we have:

$$\mathbb{E}_\gamma\left[R_T\right] \leq \sqrt{\sum_{\tau=1}^{T/m} \|\boldsymbol{x_\tau} - \tilde{\boldsymbol{x}}_{\boldsymbol{\tau}}\|_1^2} \left(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\right)$$

When we consider $x_{\tau,i} \sim Poisson(\lambda_i m)$. There are 2 options[12]:

1) prediction is a constant and a perfect predictor for expected number of future requests.

Let $C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}} = k$, then we have:

$$\begin{aligned}
\mathbb{E}_\gamma\left[R_T\right] &\leq \mathbb{E}_\gamma\left[k\sqrt{\sum_{\tau=1}^{T/m}\|\boldsymbol{x_\tau}-\tilde{\boldsymbol{x}}_{\boldsymbol{\tau}}\|_1^2}\right] \\
&= \mathbb{E}_\gamma\left[k\sqrt{\sum_{\tau=1}^{T/m}\left(\sum_{i=1}^{N}\sqrt{(x_{\tau,i}-\tilde{x}_{\tau,i})^2}\right)^2}\right] \\
&\leq k\sqrt{\sum_{\tau=1}^{T/m}\left(\sum_{i=1}^{N}\sqrt{\mathbb{E}_\gamma[(x_{\tau,i}-\tilde{x}_{\tau,i})^2]}\right)^2} \\
&= k\sqrt{\sum_{\tau=1}^{T/m}\left(\sum_{i=1}^{N}\sqrt{Var(x_{\tau,i})}\right)^2} \\
&= k\sqrt{\sum_{\tau=1}^{T/m}\left(\sum_{i=1}^{N}\sqrt{\lambda_i m}\right)^2} = k\sqrt{T\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2}
\end{aligned}$$

This means that with perfect prediction there is no loss in batched cases compared to different batch size.

2) $x_{\tau,i}, \tilde{x}_{\tau,i} \sim Poisson(\lambda_i m)$

Looking at the expection of $(x_{\tau,i}-\tilde{x}_{\tau,i})^2$, We have:

$$\begin{aligned}
&\mathbb{E}\left[(x_{\tau,i}-\tilde{x}_{\tau,i})^2\right] = \mathbb{E}\left[(x_{\tau,i}-\tilde{x}_{\tau,i}-\mathbb{E}\left[x_{\tau,i}\right]+\mathbb{E}\left[x_{\tau,i}\right]-\mathbb{E}\left[\tilde{x}_{\tau,i}\right]+\mathbb{E}\left[\tilde{x}_{\tau,i}\right])^2\right] = \\
&\mathrm{Var}\left(x_{\tau,i}\right)+\mathrm{Var}\left(\tilde{x}_{\tau,i}\right)+\left(\mathbb{E}\left[x_{\tau,i}\right]-\mathbb{E}\left[\tilde{x}_{\tau,i}\right]\right)^2 = \begin{cases} 2\lambda_i m, & \tau > 1 \\ \lambda_i m + \left(\frac{m}{m_0}\right)^2 \lambda_i m_0, & \tau = 1. \end{cases}
\end{aligned} \tag{3.10}$$

The initial prediction is given by $x_{1,i} = \frac{n_{1,i}(m_0)}{m_0}$, where $n_{\tau,i}(m)$ is the number of arrivals in the interval $[\tau, \tau+m]$, and m_0 is a first warm-up interval.

Finally we have:

$$\sum_{\tau=1}^{T/m}\left(\sum_{i=1}^{N}\sqrt{\mathbb{E}_\gamma[(x_{\tau,i}-\tilde{x}_{\tau,i})^2]}\right)^2 = \left(\sum_{i=1}^{N}\sqrt{\lambda_i m + \frac{m^2\lambda_i}{m_0}}\right)^2 + \left(\frac{T}{m}-1\right)\left(\sum_{i=1}^{N}\sqrt{2\lambda_i m}\right)^2 \tag{3.11}$$

Taking the first derivative of the above result:

$$
\begin{aligned}
&2\left(\sum_{i=1}^{N}\sqrt{\lambda_i m+\frac{m^2\lambda_i}{m_0}}\right)\left(\frac{1}{2}\sum_{i=1}^{N}\frac{\lambda_i+\frac{2m\lambda_i}{m_0}}{\sqrt{\lambda_i m+\frac{m^2\lambda_i}{m_0}}}\right)-\frac{T}{m^2}\left(\sum_{i=1}^{N}\sqrt{2\lambda_i m}\right)^2\\
&+2\left(\frac{T}{m}-1\right)\left(\sum_{i=1}^{N}\sqrt{2\lambda_i m}\right)\left(\frac{1}{2}\sum_{i=1}^{N}\frac{2\lambda_i}{\sqrt{2\lambda_i m}}\right)\\
&=\left(\sum_{i=1}^{N}\sqrt{\lambda_i m+\frac{m^2\lambda_i}{m_0}}\right)\left(\sum_{i=1}^{N}\frac{\sqrt{\lambda_i}+\frac{2m\sqrt{\lambda_i}}{m_0}}{\sqrt{m+\frac{m^2}{m_0}}}\right)-\frac{T}{m^2}\left(\sum_{i=1}^{N}\sqrt{2\lambda_i m}\right)^2\\
&+\left(\frac{T}{m}-1\right)\left(\sum_{i=1}^{N}\sqrt{\lambda_i m}\right)\left(\sum_{i=1}^{N}\frac{2\sqrt{\lambda_i}}{\sqrt{m}}\right)\\
&=\left(\sum_{i=1}^{N}\sqrt{\lambda_i+\frac{m\lambda_i}{m_0}}\right)\left(\sum_{i=1}^{N}\frac{\sqrt{\lambda_i}+\frac{2m\sqrt{\lambda_i}}{m_0}}{\sqrt{1+\frac{m}{m_0}}}\right)-\frac{T}{m}\left(\sum_{i=1}^{N}\sqrt{2\lambda_i}\right)^2+2\left(\frac{T}{m}-1\right)\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2\\
&=\sqrt{1+\frac{m}{m_0}}\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)\left(\frac{1+\frac{2m}{m_0}}{\sqrt{1+\frac{m}{m_0}}}\right)\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)-\frac{2T}{m}\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2+2\left(\frac{T}{m}-1\right)\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2\\
&=\left(1+\frac{2m}{m_0}\right)\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2-\frac{2T}{m}\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2+2\left(\frac{T}{m}-1\right)\left(\sum_{i=1}^{N}\sqrt{\lambda_i}\right)^2
\end{aligned}
$$

Let it be equal to 0, we finally have the following optimal batch size: $m_0/2$. □

When we assume that requests follow a Poisson distribution, we can see that when the prediction is a mean of future expectations, our bounds will be independent of the batch size. If we further assume that the predictions also obey the same Poisson distribution, we can get an optimal batch size $m_0/2$.

3.3 OFTPL with Switching Costs

In this section, we discuss the performance of OFTPL (Optimistic Follow the Perturbed Leader) considering the switching cost. Let's start with the definition of the switching cost: From [11], we know the switching cost can be written as follows: $\frac{D}{2}\sum_{t=1}^{T}\mathbb{E}\left\|\boldsymbol{y}_t-\boldsymbol{y}_{t-1}\right\|_1$

Let's look at the vector in the equation above:

$$
\begin{aligned}
\boldsymbol{y}_t-\boldsymbol{y}_{t-1}&\overset{(a)}{=}\nabla\Phi_t\left(\boldsymbol{X_{t-1}}+\tilde{\boldsymbol{x}}_{\boldsymbol{t}}\right)-\nabla\Phi_{t-1}\left(\boldsymbol{X_{t-1}}+\tilde{\boldsymbol{x}}_{\boldsymbol{t-1}}\right)\\
&\overset{(b)}{=}\mathbb{E}_\gamma\left[\gamma\left(\frac{1}{\eta_t}\Phi\left(\boldsymbol{X_{t-1}}+\tilde{\boldsymbol{x}}_{\boldsymbol{t}}+\eta_t\boldsymbol{\gamma}\right)-\frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X_{t-2}}+\tilde{\boldsymbol{x}}_{\boldsymbol{t-1}}+\eta_{t-1}\boldsymbol{\gamma}\right)\right)\right]\\
&=\mathbb{E}_\gamma\left[\boldsymbol{\gamma}J_t\right]
\end{aligned}
\tag{3.12}
$$

Where $J_t = \frac{1}{\eta_t}\Phi\left(\boldsymbol{X_{t-1}} + \tilde{\boldsymbol{x}}_{\boldsymbol{t}} + \eta_t\boldsymbol{\gamma}\right) - \frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X_{t-2}} + \tilde{\boldsymbol{x}}_{\boldsymbol{t-1}} + \eta_{t-1}\boldsymbol{\gamma}\right)$, $\Phi(x) = \max_{y\in\mathcal{Y}}\langle y, x\rangle$ is a baseline potential function. The first step (a) is from ([21], Theorem 3), and the second step (b) is from ([26], Lemma 7).

The relation between $\Phi_t(\cdot)$ and $\Phi(\cdot)$ is:

$$\Phi_t(\boldsymbol{x}) = \mathop{\mathbb{E}}_{\gamma\sim\mathcal{N}(0,I)}\left[\max_{\boldsymbol{y}\in\mathcal{X}}\langle \boldsymbol{y}, \boldsymbol{x} + \eta_t\boldsymbol{\gamma}\rangle\right] = \mathop{\mathbb{E}}_{\gamma}\left[\Phi\left(\boldsymbol{x} + \eta_t\boldsymbol{\gamma}\right)\right] \tag{3.13}$$

Algorithm 3 The OFTPL Caching Policy with Switching Cost

Input: $\eta_1 = 0, \boldsymbol{y_1} = \arg\min_{\boldsymbol{y}\in X}\langle \boldsymbol{y}, \boldsymbol{x_1}\rangle$
Output: $\boldsymbol{y_t} \in \mathcal{Y}$
Sample: $\boldsymbol{\gamma} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
for t=2,3... *to* T **do**
  $\tilde{\boldsymbol{x}}_{\boldsymbol{t}} \leftarrow$ prediction
  $\eta_t = \sqrt{\frac{4D}{2C\sqrt{\pi\ln(Ne/C)}}}\sqrt{(t-1) + \frac{1}{4}\sum_{\tau=1}^{t-1}\|\boldsymbol{x_\tau} - \tilde{\boldsymbol{x}}_{\boldsymbol{\tau}}\|_1^2}$
  $\boldsymbol{y}_t = \text{argmax}_{y\in X}\langle \boldsymbol{y}, \boldsymbol{X_{t-1}} + \tilde{\boldsymbol{x}}_{\boldsymbol{t}} + \eta_t\boldsymbol{\gamma}\rangle$
  The policy receives a reward $q_t = \langle \boldsymbol{y}_t, \boldsymbol{x}_t\rangle - \frac{D}{2}\|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\|_1$
  $X_t = X_{t-1} + x_t$
end for

Through the above steps, we can see that, in order to obtain the boundary of the switching cost, we need to analyze J_t first. Then we can get the result of Lemma 3.2 as follows.

Lemma 3.2. For $J_t = \frac{1}{\eta_t}\Phi\left(\boldsymbol{X}_{t-1} + \tilde{\boldsymbol{x}}_t + \eta_t\boldsymbol{\gamma}\right) - \frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X}_{t-2} + \tilde{\boldsymbol{x}}_{t-1} + \eta_{t-1}\boldsymbol{\gamma}\right)$, generally we have:

$$|J_t| \le 1 + \frac{1}{4}\|\boldsymbol{x_t} - \tilde{\boldsymbol{x}}_{\boldsymbol{t}}\|_1^2 \tag{3.14}$$

Proof. $J_t = \frac{1}{\eta_t}\Phi\left(\boldsymbol{X}_{t-1} + \tilde{\boldsymbol{x}}_t + \eta_t\boldsymbol{\gamma}\right) - \frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X}_{t-2} + \tilde{\boldsymbol{x}}_{t-1} + \eta_{t-1}\boldsymbol{\gamma}\right)$.

Let us consider two cases, 1) $J_t > 0$, 2) $J_t \le 0$.

1) $J_t > 0$

$$
\begin{aligned}
|J_t| &= \frac{1}{\eta_t}\Phi\left(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t+\eta_t\boldsymbol{\gamma}\right)-\frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1}+\eta_{t-1}\boldsymbol{\gamma}\right)\\
&\overset{(a)}{=} \Phi\left(\frac{1}{\eta_t}(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t+\eta_t\boldsymbol{\gamma})\right)-\Phi\left(\frac{1}{\eta_{t-1}}(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1}+\eta_{t-1}\boldsymbol{\gamma})\right)\\
&\overset{(a)}{\leq} \Phi\left(\frac{1}{\eta_t}(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t)-\frac{1}{\eta_{t-1}}(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1})\right)\\
&\overset{(b)}{\leq} \frac{1}{\eta_t}\Phi\left(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t-\boldsymbol{X}_{t-2}-\tilde{\boldsymbol{x}}_{t-1}\right)\\
&= \frac{1}{\eta_t}\Phi\left(\boldsymbol{x}_{t-1}+\tilde{\boldsymbol{x}}_t-\tilde{\boldsymbol{x}}_{t-1}\right)\\
&= \frac{1}{\eta_t}\left(1+\frac{1}{4}\|\boldsymbol{x}_{t-1}-\tilde{\boldsymbol{x}}_{t-1}\|_1^2\right)
\end{aligned}
$$

where (a) is the property of sublinear function, (b) represents the monotonically increasing property of the function. In the last step, $\Phi(x_{t-1}+\tilde{x}_t-\tilde{x}_{t-1})$ is 1 when the prediction is right and 2 when it is not. $\Phi\left(x_{t-1}+\tilde{x}_t-\tilde{x}_{t-1}\right)$ has the same value in single cache problem.

Finally we can have the final equation hold, which means we can bound J_t by using numbers and prediction accuracy, and it would be helpful for the following proof.

2) $J_t \leq \mathbf{0}$

$$
\begin{aligned}
|J_t| &= |\frac{1}{\eta_t}\Phi\left(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t+\eta_t\boldsymbol{\gamma}\right)-\frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1}+\eta_{t-1}\boldsymbol{\gamma}\right)|\\
&= \frac{1}{\eta_{t-1}}\Phi\left(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1}+\eta_{t-1}\boldsymbol{\gamma}\right)-\frac{1}{\eta_t}\Phi\left(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t+\eta_t\boldsymbol{\gamma}\right)\\
&\overset{(a)}{=} \Phi\left(\frac{1}{\eta_{t-1}}(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1}+\eta_{t-1}\boldsymbol{\gamma})\right)-\Phi\left(\frac{1}{\eta_t}(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t+\eta_t\boldsymbol{\gamma})\right)\\
&\overset{(a)}{\leq} \Phi\left(\frac{1}{\eta_{t-1}}(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1})-\frac{1}{\eta_t}(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t)\right)\\
&= \frac{1}{\eta_t}\Phi\left(\frac{\eta_t}{\eta_{t-1}}(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1})-(\boldsymbol{X}_{t-1}+\tilde{\boldsymbol{x}}_t)\right)\\
&\overset{(a)}{\leq} \frac{1}{\eta_t}\left[\Phi(\frac{\eta_t}{\eta_{t-1}}\boldsymbol{X}_{t-2}-\boldsymbol{X}_{t-2})+\Phi(\frac{\eta_t}{\eta_{t-1}}\tilde{\boldsymbol{x}}_{t-1}-\boldsymbol{x}_{t-1}-\tilde{\boldsymbol{x}}_t)\right]\\
&= \frac{1}{\eta_t}\left[\frac{\eta_t-\eta_{t-1}}{\eta_{t-1}}\Phi(\boldsymbol{X}_{t-2})+\Phi(\frac{\eta_t}{\eta_{t-1}}\tilde{\boldsymbol{x}}_{t-1}-\boldsymbol{x}_{t-1})\right]\\
&\overset{(a)}{\leq} \frac{1}{\eta_t}\left[\frac{\eta_t-\eta_{t-1}}{\eta_{t-1}}\Phi(\boldsymbol{X}_{t-2})+\Phi(\tilde{\boldsymbol{x}}_{t-1}-\boldsymbol{x}_{t-1})+\Phi(\frac{\eta_t-\eta_{t-1}}{\eta_{t-1}}\tilde{\boldsymbol{x}}_{t-1})\right]\\
&\overset{(a)}{\leq} \frac{1}{\eta_t}\left[\frac{\eta_t-\eta_{t-1}}{\eta_{t-1}}\Phi(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1})+\Phi(\tilde{\boldsymbol{x}}_{t-1}-\boldsymbol{x}_{t-1})\right]
\end{aligned}
$$

Where (a) is the property of sublinear function. We also have

$$
\frac{\eta_t-\eta_{t-1}}{\eta_{t-1}}\Phi(\boldsymbol{X}_{t-2}+\tilde{\boldsymbol{x}}_{t-1}) \leq \frac{\eta_t-\eta_{t-1}}{\eta_{t-1}}(t-1) \tag{3.15}
$$

with the definition of $\Phi(\cdot)$. If we choose $\eta_t = \beta\sqrt{\sum_{i=1}^{t-1}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}$, we have:

$$\begin{aligned}\frac{\eta_t - \eta_{t-1}}{\eta_{t-1}}(t-1) &= \frac{\sqrt{\sum_{i=1}^{t-1}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)} - \sqrt{\sum_{i=1}^{t-2}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}}{\sqrt{\sum_{i=1}^{t-2}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}}(t-1) \\ &= \frac{1+\frac{1}{4}\|\boldsymbol{x}_{t-1} - \tilde{\boldsymbol{x}}_{t-1}\|_1^2}{\frac{\sqrt{\sum_{i=1}^{t-1}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}\sqrt{\sum_{i=1}^{t-2}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}}{t-1} + \frac{\sum_{i=1}^{t-2}(1+\frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}{t-1}}\end{aligned} \tag{3.16}$$

Let's discuss Equation (3.16) further:

1) prediction is always right.

In this case, we have:

$$\frac{\eta_l - \eta_{l-1}}{\eta_{l-1}}(t-1) = \frac{1}{\frac{\sqrt{(t-1)(t-2)}}{t-1} + \frac{t-2}{t-1}}$$

We can see that it is always less than 1 for any $t > 2$.

2) there is sum of prediction error e in the past, where $e \geq 1$.

In this case, the denominator we can write as:

$$\text{denominator of Equation (3.16)} = \frac{\sqrt{(t-1+e)(t-2+e)}}{t-1} + \frac{t-2+e}{t-1}$$

We can ensure that the denominator is a positive number greater than 2, with an $e \geq 1$, which means in this case it is still a number less than 1.

3) past predictions are right, and in time $t-1$ there is a prediction error.

This is a special case that occurs only once. Before this, all possible cases are 1), after this will be transferred to 2). And we can find:

$$\frac{\eta_l - \eta_{l-1}}{\eta_{l-1}}(t-1) = \frac{2}{\frac{\sqrt{(t)(t-2)}}{t-1} + \frac{t-2}{t-1}}$$

It does not hold that it is less than 1 when t is small, but it is smaller than 2 in any case. Since it can only appear once, we can add the part greater than 1 to the final bound as a constant item to make the above conclusions consistent.

Finally, we can say that $\frac{\eta_t - \eta_{t-1}}{\eta_{t-1}}(t-1) \leq 1$. (it may be greater than 1 (< 2) only when the first prediction error occur. We can add 1 into our final sum of regret to

remove this approximate error). Generally, $|J_t| \leq \frac{1}{\eta_t}(1 + \frac{1}{4}\|x_{t-1} - \tilde{x}_{t-1}\|_1^2)$ is hold from above results □

Now that we have the upper bound of J_t, we can then analyze the initial boundary to get the upper bound of the final switching cost.

Lemma 3.3. Switching cost in OFTPL case $\leq \frac{2D}{\beta\sqrt{2\pi}}\sqrt{\sum_{\tau=1}^{T}(4 + \|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)} + c_0$ where c_0 is a constant smaller than $D\sqrt{\frac{2}{\pi}}$.

Proof. By the definition of switching cost and results from Lemma 3.2, if there is 1 switch at most in one step we have:

$$\begin{aligned}
\frac{D}{2}\sum_{t=1}^{T}\mathbb{E}\left\|\boldsymbol{y}_t - \boldsymbol{y}_{t-1}\right\|_1 &\leq D\sum_{t=1}^{T}\left[\frac{1}{\eta_t}\sqrt{\frac{2}{\pi}}(1 + \frac{1}{4}\|\boldsymbol{x}_{t-1} - \tilde{\boldsymbol{x}}_{t-1}\|_1^2)\right] \\
&= \frac{D}{\beta}\sqrt{\frac{2}{\pi}}\sum_{t=1}^{T}\left[\frac{1 + \frac{1}{4}\|\boldsymbol{x}_{t-1} - \tilde{\boldsymbol{x}}_{t-1}\|_1^2}{\sqrt{\sum_{i=1}^{t-1}(1 + \frac{1}{4}\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}}\right] \\
&= \frac{D}{\beta\sqrt{2\pi}}\sum_{t=1}^{T}\left[\frac{4 + \|\boldsymbol{x}_{t-1} - \tilde{\boldsymbol{x}}_{t-1}\|_1^2}{\sqrt{\sum_{i=1}^{t-1}(4 + \|x_i - \tilde{\boldsymbol{x}}_i\|_1^2)}}\right] \\
&\leq \frac{D}{\beta\sqrt{2\pi}}\sum_{t=1}^{T}\left[\frac{4 + \|\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\|_1^2}{\sqrt{\sum_{i=1}^{t}(4 + \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|_1^2)}}\right]
\end{aligned}$$

Since:

$$\begin{aligned}
\sum_{t=1}^{T}\frac{4 + \|\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t\|_1^2}{\sqrt{\sum_{\tau=1}^{t}(4 + \|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)}} &\leq \sum_{t=1}^{T}\int_{\sum_{\tau=1}^{t-1}(4+\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)}^{\sum_{\tau=1}^{t}(4+\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)}\frac{dx}{\sqrt{x}} \\
&= \int_0^{\sum_{\tau=1}^{T}(4+\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)}\frac{dx}{\sqrt{x}} \\
&= 2\sqrt{\sum_{\tau=1}^{T}(4 + \|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)}
\end{aligned} \tag{3.17}$$

Finally, we have:

$$\text{switching cost} \leq \frac{2D}{\beta\sqrt{2\pi}}\sqrt{\sum_{\tau=1}^{T}(4+\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2)}+\textit{estimated error}(c_0) \tag{3.18}$$

□

When we get the bound of the exchange cost in this case, we can simply get the final upper bound of regret by combining the part that does not consider the switching cost (Theorem 3.3).

Theorem 3.3. The regret bound of OFTPL with switching cost:

$$\begin{aligned}\mathbb{E}[R_T] \leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2)}+\frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}\\ +\frac{4D}{\beta\sqrt{2\pi}}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2)}+c_0\end{aligned} \tag{3.19}$$

Proof. From [21], we can get the regret of OFTPL with our parameter.

$$\mathop{\mathbb{E}}_{\gamma}[R_T] \leq \eta_T C\sqrt{2\ln(Ne/C)}+\sum_{t=1}^{T}\frac{1}{2}\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_{H_t} \tag{3.20}$$

With our η, we have:

$$\begin{aligned}\frac{1}{2}\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_{H_t} &\leq \min\left(\frac{1}{\sqrt{2\pi}}\frac{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}{\eta_t}, 2\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1\right)\\ &= \min\left(\frac{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}{\sqrt{2\pi}\beta\sqrt{1+\sum_{\tau=1}^{t-1}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}}, 2\frac{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}{\sqrt{2\pi}\beta\sqrt{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}}\right)\\ &\leq \min\left(\frac{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}{\sqrt{2\pi}\beta\sqrt{\sum_{\tau=1}^{t-1}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}}, 2\frac{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}{\sqrt{2\pi}\beta\sqrt{\|\boldsymbol{x}_t-\tilde{\boldsymbol{x}}_t\|_1^2}}\right)\end{aligned} \tag{3.21}$$

Using the same way ([21], Theorem 3), we can get part of the final regret bound

without switching cost:

$$\mathop{\mathbb{E}}_{\gamma}[R_T] \leq \eta_T C\sqrt{2\log(Ne/C)} + \frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T}\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2}. \tag{3.22}$$

With switching cost (c_0 is the constant from the error in the first part, it should be less than $D\sqrt{\frac{2}{\pi}}$):

$$\begin{aligned}\mathbb{E}[R_T] &\leq \eta_T C\sqrt{2\ln(Ne/C)} + \frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T}\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2} + \frac{4D}{\beta\sqrt{2\pi}}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)} + c_0 \\ &\leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)} + \frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T}\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2} \\ &+ \frac{4D}{\beta\sqrt{2\pi}}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2)} + c_0\end{aligned} \tag{3.23}$$

□

Through the above proof, we get the upper bound of the OFTPL algorithm with the switching cost. We can see that this boundary is sublinear. Next let's discuss how our bound performs compared to algorithms without predictions.

First, we use Lemma 3.4 to compare the impact of different learning rates η on the boundary without prediction. Because different forms of learning rates will get different bounds, we need to choose a suitable bound for comparison.

Lemma 3.4. Without predictions, the regret obtained by the dynamic learning rate is greater than the fixed learning rate.

Proof. From [11], we have 2 bound for FTPL with switching cost:

- Fixed η: $\mathbb{E}(R_T) \leq C\eta\sqrt{2\ln(N/C)} + \frac{T}{\eta\sqrt{2\pi}}(1+D)$, with $\eta = \beta\sqrt{T}$.
- Dynamic η ($\eta_t = \alpha\sqrt{t}$): $\mathbb{E}(R_T) \leq c_1\sqrt{T} + c_2\ln T + c_3$, with $c_1 = \alpha C\sqrt{2\ln(Ne/C)} + \frac{(2+3D)}{\alpha\sqrt{2\pi}}$

For the convenience of analysis, we simplify the regret with the dynamic learning

rate as follows:

$$\mathbb{E}(R_{T,dynamic}) \leq \left(\alpha C\sqrt{2\ln(Ne/C)} + \frac{(2+3D)}{\alpha\sqrt{2\pi}}\right)\sqrt{T} \tag{3.24}$$

In fact, when we consider reasonable N and T, our simplified bound is always smaller than the original bound.

Now let us look at the fixed η case:

$$\mathbb{E}(R_{T,fixed}) \leq \left(C\beta\sqrt{2\ln(N/C)} + \frac{1+D}{\beta\sqrt{2\pi}}\right)\sqrt{T} \tag{3.25}$$

We can easily see that when the learning rate changes over time, the regret is always greater than one with the fixed learning rate, because at least we can choose $\beta = \alpha$ for any α and we will get this conclusion. This may be due to the fact that the fixed learning rate has more information. □

We should compare our algorithm performance with dynamic learning rate, for OFTPL with switching cost has a dynamic learning rate. It can be seen that when there are good predictions, we can get a better bound.

Lemma 3.5. In the case of dynamic learning rates, considering perfect predictions with switching cost, we will get a better regret bound than one without predictions.

Proof. We still consider a simplified one for case without prediction:

$$\mathbb{E}(R_{T,dynamic}) \leq \left(\alpha C\sqrt{2\ln(Ne/C)} + \frac{(2+3D)}{\alpha\sqrt{2\pi}}\right)\sqrt{T}$$

With perfect predictions, the bound of regret we have is (without constant c_0):

$$\mathbb{E}[R_{T,pred}] \leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{T} + \frac{4D}{\beta\sqrt{2\pi}}\sqrt{T} \tag{3.26}$$

We can see that in the case of dynamic learning rate, with perfect predictions, it can get a smaller bound for the regret if $D \leq 2$. □

Comparing the two optimal parameters, we have $\alpha^* = \sqrt{\frac{2+3D}{2C\sqrt{\pi\ln(Ne/C)}}}$ and $\beta^* =$

$\sqrt{\frac{4D}{2C\sqrt{\pi \ln(Ne/C)}}}$. With $D \geq 2$, it means $\beta^* \geq \alpha^*$, it will learn more from noise. Since our noise does not change after the initial sampling, a larger noise learning rate means that the predictions will be closer to the noise, that is, the switching will not tend to occur after a long learning period, because most of the predictions will fall in the files with larger noise term.

From Lemma 3.5 we can choose an optimistic $\beta_{opt} = \sqrt{\frac{2D}{C\sqrt{\pi \ln(Ne/C)}}}$, which can ensure a smaller regret with good predictions. Substituting the β_{opt} we obtained into original regret bound Equation (3.19) (without c_0), we have:

$$\mathbb{E}[R_{T,pred}] \leq 4\sqrt{DC}\pi^{-\frac{1}{4}} \ln(Ne/C)^{\frac{1}{4}}\sqrt{T} + D\sqrt{\frac{2}{\pi}} \tag{3.27}$$

The above results mean that when we choose an optimistic β, when we consider a case where the switching cost is small enough ($0 < \beta \leq \frac{1}{\sqrt{2\pi}}$), we can get a bound close to 0.

Let's go back to the beginning and consider the case where the switching cost is 0 with perfect prediction. At this time, our regret boundary is $\mathbb{E}[R_T, pred] \leq \eta_T C\sqrt{2\ln(Ne/C)} + \frac{1}{2}\left\|\theta_t - \tilde{\theta}_t\right\|_{H_t} = \eta_T C\sqrt{2\ln(Ne/C)}$. We can choose $\eta_T = 0$ to get a 0 regret bound. To sum up, when there are perfect predictions, we can get a smaller regret bound as the switching cost decreases until both parts are equal to 0.

We discussed the case of having perfect predictions, then let us analyze how our bounds change when the predictions are not perfect. Let's define a new parameter ϵ, the accuracy of prediction. Based on this definition, we have $\sum_{\tau=1}^{T} \|\boldsymbol{x}_\tau - \tilde{\boldsymbol{x}}_\tau\|_1^2 = 4T \times (1-\epsilon)$. Then we can get Lemma 3.6.

Lemma 3.6. For the FTPL algorithm, with or without switching cost, in order to get a better theoretical bound in the case of prediction, a high prediction accuracy is required.

Proof. 1) without switching cost.

If we use the accuracy defined before, we can have the bound below:

- without predictions, $\mathbb{E}(R_T) \leq C\eta\sqrt{2\ln(N/C)} + \frac{T}{\eta\sqrt{2\pi}}$

- with predictions, $\mathbb{E}\left[R_T, pred\right] \leq 2\sqrt{(1-\epsilon)T}(C\sqrt{2\ln(Ne/C)})\beta + \frac{6}{\beta\sqrt{2\pi}}\Big)$

By comparing these two boundaries, we can get that the prediction accuracy rate must be at least greater than 95%, then we can get a better bound.

2) with switching cost

Similarly, we have the following bounds:

- $\mathbb{E}(R_T) \leq \left(\alpha C\sqrt{2\ln(Ne/C)} + \frac{(2+3D)}{\alpha\sqrt{2\pi}}\right)\sqrt{T}$
- $\mathbb{E}\left[R_{T,pred}\right] \leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{2-\epsilon} + \frac{6}{\sqrt{2\pi}\beta}\sqrt{1-acc} + \frac{4D}{\beta\sqrt{2\pi}}\sqrt{2-\epsilon}$

By comparing these two boundaries, we can get that the prediction accuracy rate must be at least greater than 97% in this case for any D, otherwise we cannot get a better bound. □

Through Lemma 3.6, we may find that it requires a really high accuracy to consider predictions. It looks confused, but we need remember the regret is the upper bound of the loss compared to the best static strategy. It just means in worst cases, we really need a high accuracy to have a better bound. In most cases, the algorithm will perform better with predictions that are not too low in accuracy.

3.4 OFTPL with Batched Requests and Switching Costs

Based on the above work, we have obtained the regret of OFTPL with switching cost. In this part we discuss the regret of batched requests for OFTPL with switching cost, and we also assume that the batch size is m.

Theorem 3.4. For OFTPL with switching cost and batch m, the regret is:

$$\mathbb{E}[R_T] \leq \left(\beta C\sqrt{2\ln(Ne/C)}\sqrt{m+1} + \frac{12}{\sqrt{2\pi}\beta}\sqrt{m} + \frac{4D}{\beta\sqrt{2\pi}}\sqrt{m+1}\right)\sqrt{T} \tag{3.28}$$

Proof. From Equation (3.19), we have:

$$\mathbb{E}[R_T] \leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2)}+\frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}$$
$$+\frac{4D}{\beta\sqrt{2\pi}}\sqrt{\sum_{\tau=1}^{T}(1+\frac{1}{4}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2)}+c_0$$

In batched requests,

$$\mathbb{E}[R_T] \leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{T+\sum_{\tau=1}^{T/m}\frac{1}{4}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}+\frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T/m}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}$$
$$+\frac{4D}{\beta\sqrt{2\pi}}\sqrt{T+\sum_{\tau=1}^{T/m}\frac{1}{4}\|\boldsymbol{x}_\tau-\tilde{\boldsymbol{x}}_\tau\|_1^2}+c_0$$
$$\leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{T+\sum_{\tau=1}^{T/m}\frac{1}{4}\left(\sum_{i=1}^{N}|x_{\tau,i}-\tilde{x}_{\tau,i}|\right)^2}+\frac{6}{\sqrt{2\pi}\beta}\sqrt{\sum_{\tau=1}^{T/m}\left(\sum_{i=1}^{N}|x_{\tau,i}-\tilde{x}_{\tau,i}|\right)^2}$$
$$+\frac{4D}{\beta\sqrt{2\pi}}\sqrt{T+\sum_{\tau=1}^{T/m}\frac{1}{4}\left(\sum_{i=1}^{N}|x_{\tau,i}-\tilde{x}_{\tau,i}|\right)^2}+c_0$$
$$\leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{T+mT}+\frac{6}{\sqrt{2\pi}\beta}\sqrt{4mT}+\frac{4D}{\beta\sqrt{2\pi}}\sqrt{T+mT}+c_0$$
$$=\left(\beta C\sqrt{2\ln(Ne/C)}\sqrt{m+1}+\frac{12}{\sqrt{2\pi}\beta}\sqrt{m}+\frac{4D}{\beta\sqrt{2\pi}}\sqrt{m+1}\right)\sqrt{T}.$$

□

We can see that the regret, in this case, is also sublinear with the batch size m. We can also use the method in Lemma 3.1 to make an assumption about the distribution of request and prediction sequences, and then we can get Lemma 3.7.

Lemma 3.7. Assume $x_{\tau,i} \sim Poisson(\lambda_i m)$, m_0 is a first warm-up interval. we have 2 conclusions in OFTPL with switching cost:

- with perfect predictor, batch size does not affect our regret.
- consider predictor as a random variable, we have the optimal batch size $m_0/2$.

Proof. Let's follow the proedure in Lemma 3.1.

1) prediction is a constant and a perfect predictor for expected number of future reques

In this case, we can also get:

$$\begin{aligned}\mathbb{E}[R_T] \leq \beta C\sqrt{2\ln(Ne/C)}\sqrt{T + T(\sum_{i=1}^{N}\sqrt{\lambda_i})^2} + \frac{6}{\sqrt{2\pi}\beta}\sqrt{T(\sum_{i=1}^{N}\sqrt{\lambda_i})^2} \\ + \frac{4D}{\beta\sqrt{2\pi}}\sqrt{T + T(\sum_{i=1}^{N}\sqrt{\lambda_i})^2} + c_0\end{aligned} \tag{3.29}$$

We can also see that its regret is independent of the batch size m in this case.

2) $x_{\tau,i}, \tilde{x}_{\tau,i} \sim Poisson(\lambda_i m)$

In this case, after we take the first derivative of regret, we will find that the problem we face is the same as the one in second part of Lemma 3.1, that is, we can only derive the part containing the first order module to obtain the optimal value. Based on this, we can get the same optimal batch size $m_0/2$ in this case. □

Based on the above analysis, we can see that, in the case of batched requests, we can get similar results to the previous ones for OFTPL with switching costs. That is, the batch size generally does not have a large impact on our regret bounds. Even in some ideal cases, it does not have an effect on the regret boundary.

In the next section, I will simulate and compare the performance of FTPL and OFTPL in various situations, and finally use real request data to observe the performance of Algorithms.

Chapter 4

Numerical Experiments

4.1 Experiments Setting

In our simulations, the unspecified sequences are all generated by the Zipf distribution. The parameter is α, the default value is 1.2. Number of catalogs is N =100. For the comparison of different algorithms we set the time horizon T = 10000. At each individual time t, we receive a single request. We limit the cache size to 20 units. The default value of switching cost for one unit file is 1 ($D = 1$). Note that in this experiment we assume a single-cache system.

Batch Size: When we compare the performance of batch algorithms, there is a batch size parameter m. We choose $m \in \{1, 5, 10\}$ in comparative experiments.

Predictions: We need predictions in the optimistic FTPL algorithm. The perfect prediction is the same as the requests. By default, the predictions will be generated by the same distribution of requests. For perfect predictions, they are same as requests

Metrics: In order to compare the actual performance of the algorithms, the metric we choose is the *Normalized average cost*. It is obtained by dividing the accumulated cost by the current time. When we consider the cost of switching, the cumulative cost should additionally consider whether there is a switching file in each iteration. For batched case, it should be normalized also by the batch size.

Algorithms: In the initial comparison, we mainly use LRU, LFU as comparisons to illustrate the performance of FTPL. In the follow-up batch experiments and when considering predictions and switching costs, we only compare different versions of

FTPL algorithm varying different predictions and batch size.

Parameters not mentioned in the above settings, as well as possible changes, will be introduced in each part of the results.

4.2 Results

4.2.1 Comparison of different algorithms

In this part, we mainly compare the average cost performance of LFU, LRU and FTPL without considering predictions and switching cost, and batch size m = 1. We use a Zipf distribution with parameter 1.2 to generate the request sequence and the predicted sequence in OFTPL. The result is shown in Figure 4.1.

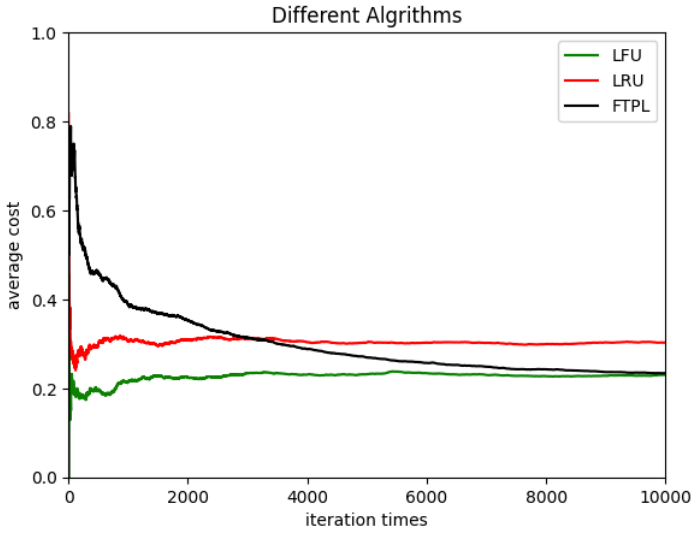

Figure 4.1. Performance comparison of different algorithms

In the case of a stable distribution of requests, LFU is theoretically the algorithm with the best performance. We can see that FTPL's performance can be close to LFU after some iterations, and at the same time, it will perform better than LRU after it has accumulated some historical information.

Next, we compare the performance of OFTPL and FTPL, and the results are shown in Figure 4.24.2. The prediction with low accuracy is to regenerate a prediction using the requested distribution, and the perfect prediction is to input the current request as the predicted value.

When we regenerated a prediction using the same distribution, it was only about 10% accurate in our setting. From the results, it can be seen that when the prediction accuracy is low (when we use the predictions generated by the same Zipf distribution again), it will have a bad impact on our final performance, and when we have a high prediction accuracy, its final performance will be better than algorithm that of not consider predictions. When predictions are perfect, it means in any time t, prediction is equal to request.

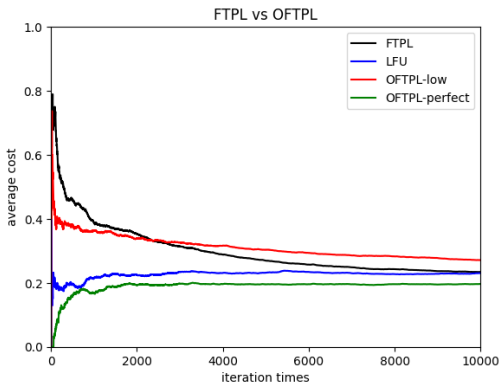

Figure 4.2. FTPL vs OFTPL

4.2.2 Batched requests

In this part, we mainly discuss the performance of the two cases we analyzed previously, namely, no prediction with exchange cost and prediction without exchange cost, under batched requests.

From Figure 4.3 We can see that as the batch size increases, our average cost also increases slightly, but behaves similarly in later stages, which matches Theorem 3.1.

When we consider the OFTPL algorithm without switching costs, the results obtained are shown in Figure 4.4. In this part, we use perfect predictions in this example. We can see that we can still get the above conclusion. At the beginning, different batch sizes will have a large difference, but the average cost will approach the same value in the later stage.

From the analysis of the above two cases, the FTPL-based online caching algorithm can still maintain a good performance in the case of batch processing.

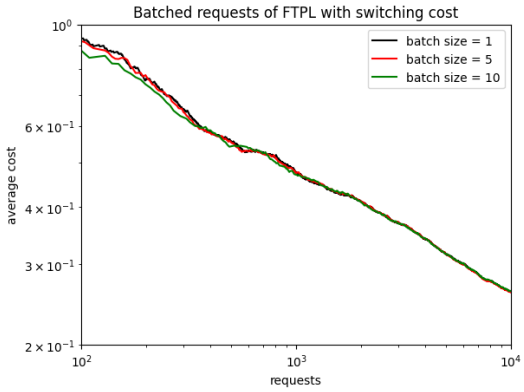

Figure 4.3. Batched requests of FTPL with switching cost

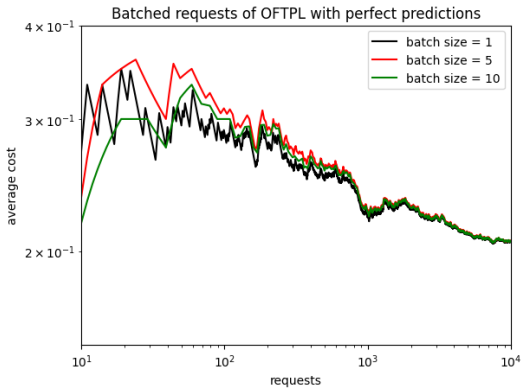

Figure 4.4. Batched requests of OFTPL without switching cost

4.2.3 OFTPL with switching cost

In this section, we will discuss the performance of OFTPL with switching costs. First let's look at how the algorithm performs with different types of learning rates. From Lemma 3.4 we know that there are two types of learning rates for FTPL that do not consider predictions, one is a fixed learning rate and the other is a learning rate that changes dynamically over time. In Figure 4.5, we compare the average cost of the above two cases, and add the OFTPL algorithm that also considers the switching cost for the dynamic learning rate.

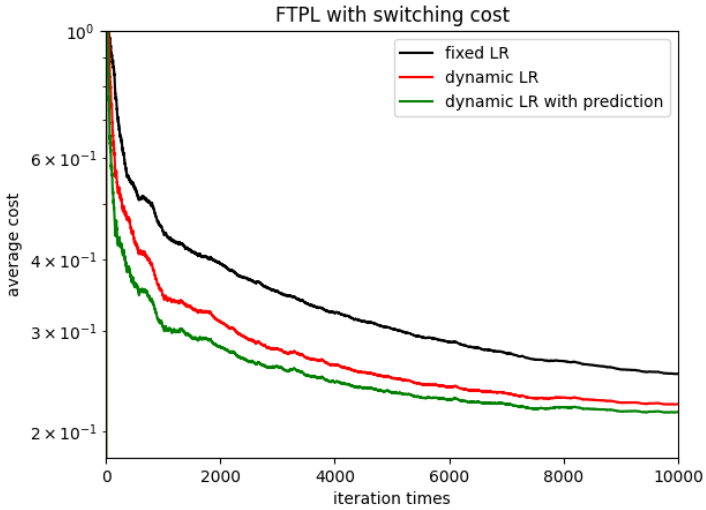

Figure 4.5. FTPL with different learning rate and OFTPL

From the results we can see that the dynamic learning rate can perform better in our example. This is because the dynamic learning rate can retain more data features at the beginning, so that the average cost can be smaller quickly. But pay attention to our conclusion in Lemma 3.4. First of all, the fixed learning rate knows more information in the algorithm. At the same time, if the requests we face are mainly noise-based at the beginning, the request distribution will be more stable in the second half, in this case their performance may be reversed.

In addition to the comparison of different types of learning rates, we can also see that the OFTPL algorithm performs better when we consider perfect predictions.

Next let us consider how our algorithms perform differently when faced with a larger switching cost. From Figure 4.5 we can clearly see the performance difference between FTPL and OFTPL (D=1). But when we choose a large exchange cost, the result is shown in Figure 4.6. We find that the performance of these two algorithms is very close at this time, even if OFTPL has a perfect prediction, and even FTPL has a lower average cost most of the time, at the end of iterations these two converge to the same.

The above results confirm the results obtained in our Lemma 3.5. But when we consider the larger switching costs, the actual situation may be more complicated. The large switching cost makes the algorithm with predictions learn more from the noise. At the same time, because there is only one-step prediction, it is often unable to make correct decision for future actions when there is a large switching cost.

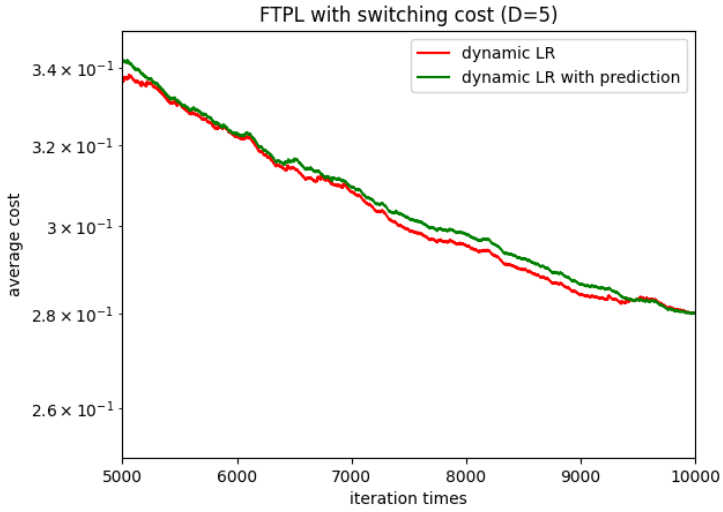

Figure 4.6. FTPL and OFTPL with D = 5

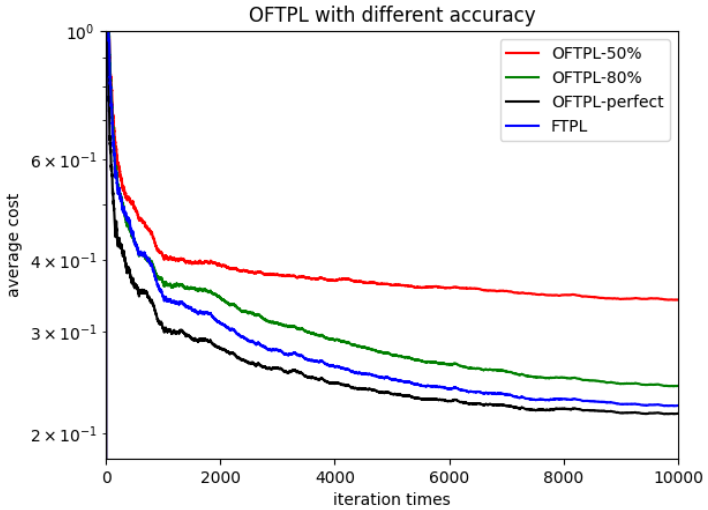

Figure 4.7. OFTPL with different prediction accuracy

In the last part, we mainly discuss how our OFTPL algorithm performs when it faces predictions with different accuracy rates. The results are shown in Figure 4.7. We can see that with perfect prediction we get better results than without prediction, but when we reduce the accuracy of the prediction, its performance will deteriorate. This is consistent with the results we obtained in Lemma 3.6.

This seems to further illustrate that we need not consider a less accurate forecast. But we note that in this example our requests are generated by the standard distribution, and incorrect predictions will make our cumulative distribution of requests

deviate from the distribution that generates requests. When the distribution of requests is more random, it is possible to consider predictions, although the accuracy rate may not be so high.

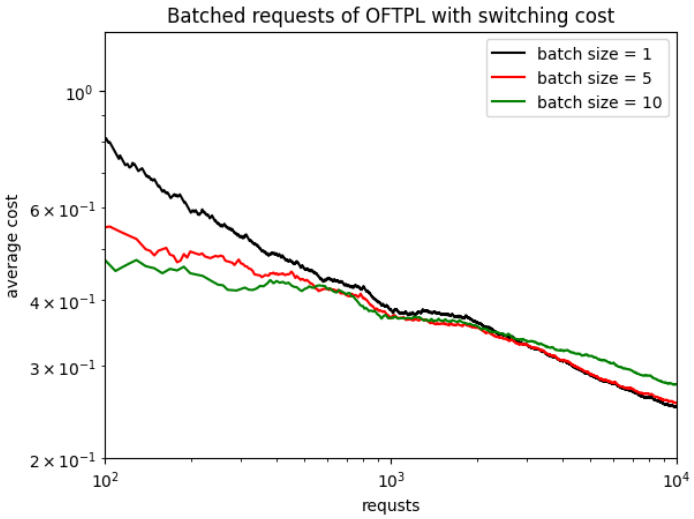

Figure 4.8. OFTPL with switching for different batch size

In Figure 4.8 we see that different batched requests still perform well in this case, and we still use the Zipf distribution. As the time increases, their average costs gradually approached the same level. Of course lower batch size will have better performance. This further confirms the feasibility of batched requests in this method.

4.2.4 Experiment with CDN data

In this section we choose to use real request data from Akamai [29] to further verify our algorithm. We use the requests data which length = 10000 during one week, and size of catalog = 500. In this part of simulation experiment, we set the number of catalog in previous part from 100 to 500. From Figure 4.9 we can see that the actual data we use also roughly follows the Zipf distribution, with only a few files being requested multiple times.

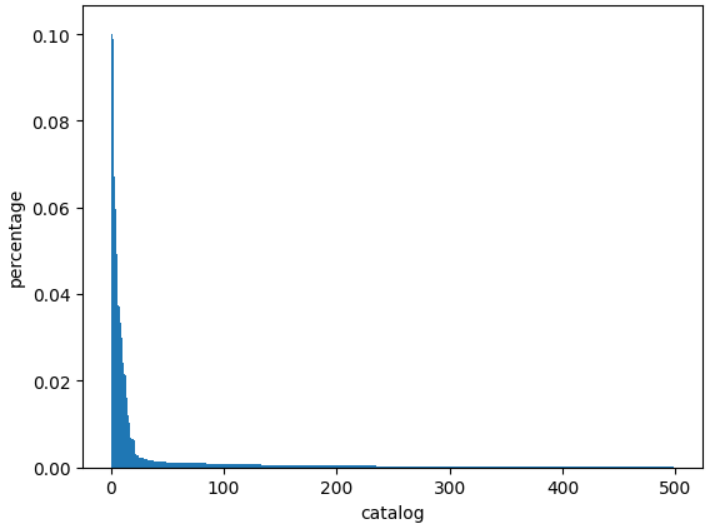

Figure 4.9. Distribution of Akamai data

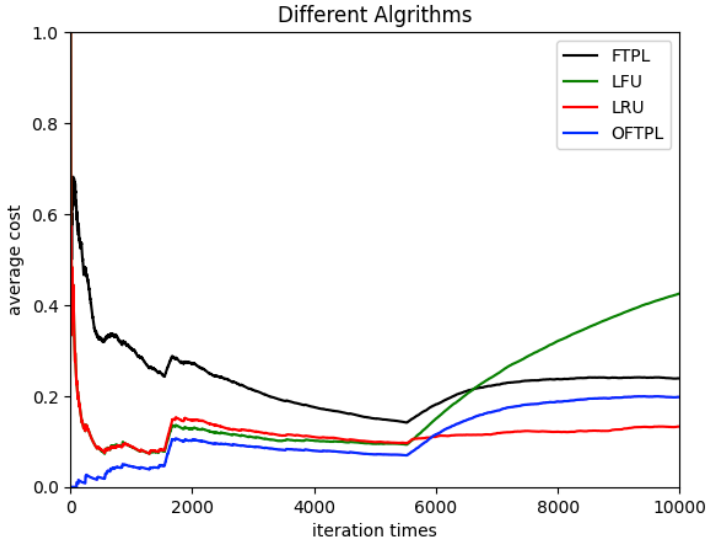

Figure 4.10. Average cost of different algorithms (without switching cost)

After observing the data characteristics of real requests, we analyzed the performance of different algorithms under this request sequence, and the results are shown in Figure 4.10. From this figure, we can see that the FTPL (OFTPL) algorithm overcomes the problem for which LFU accumulates outdated information. At the same time, the algorithm with prediction will perform better. We can also find that the LRU performs better and is very stable in the later stages of the simulation. This could be attributed to the requests becoming increasingly random; however, in scenarios with completely random requests, LRU performs well.

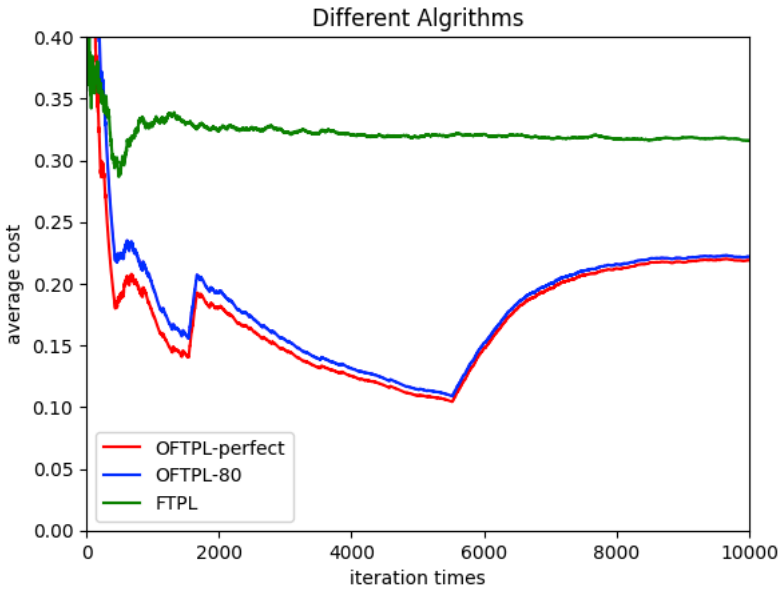

Figure 4.11. FTPL and OFTPL with different accuracy (with switching cost)

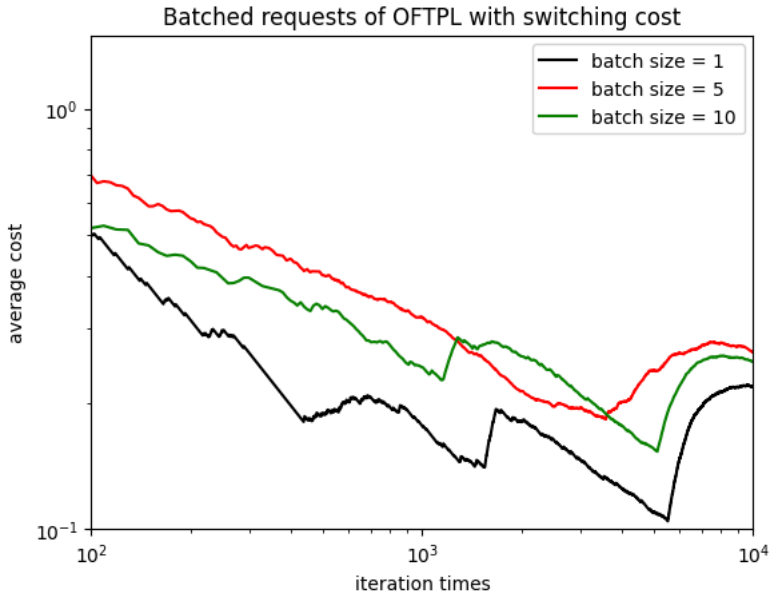

Figure 4.12. OFTPL in batched requests (with switching cost)

In the following simulations, we can see from Figure 4.11 that the algorithm with predictions performs much better, which is different from our previous results. Of course, what we get in the theoretical part is only a mathematical boundary in the worst case. In other cases there are not bad predictions, even if they do not match

the request highly, we can still get better performance. This simulation result also illustrates that in most real cases, involving predictions could get better results.

In the last part Figure 4.2, we show the performance of batched with switching costs with perfect prediction for different batch sizes. We can see that the performance is better when the batch size is smaller. At the same time, the performance gap between different batch sizes is also consistent with our previous conclusion.

4.3 Summary

In this section, we presented a comprehensive set of simulation results and analyses to evaluate the performance of various caching algorithms under different scenarios. We began by comparing LFU, LRU, and FTPL algorithms under default settings, revealing that FTPL can approach LFU's performance after some iterations and outperforms LRU over time.

We then delved into the impact of predictions, both perfect and imperfect, on the performance of OFTPL and FTPL. With low prediction accuracy, the algorithms' performance suffered, while high prediction accuracy improved results, illustrating the importance of accurate predictions. The batched requests scenario was explored next, demonstrating that the performance of FTPL-based algorithms remains favorable even with varying batch sizes. The inclusion of switching costs in OFTPL was investigated, showing that dynamic learning rates performed better than fixed rates. The influence of larger switching costs revealed change in FTPL and OFTPL performance. The experiment also highlighted the sensitivity of the performance to prediction accuracy.

Finally, a real-world scenario was examined using CDN data from Akamai. The simulation demonstrated how the algorithms perform under more complex and realistic request distributions. Overall, these simulations provided valuable insights into the performance characteristics of different caching algorithms across a range of settings, highlighting the effectiveness of FTPL-based approaches in various scenarios while emphasizing the significance of predictions, batch size and switching cost.

Chapter 5

Conclusions

In this work, we mainly discuss the performance of the FTPL algorithm under batch requests and consider predictions and switching costs in online caching problem.

First, we analyzed the version of FTPL working with batched requests and with switching cost but without considering predictions. We find that its switching cost component does not increase with batch size, and only the loss of the user's utility part will increase linearly with the increase of the exchange cost. We can see through experiments that large batch sizes incur greater costs at the initial stage, but as requests accumulate, the performance tends to approach that of single request in our case.

In the second part we continue to discuss the performance of OFTPL without considering the switching cost. Our preliminary analysis found that the regret bound increases sublinearly with the increasing of batch size. When we add assumptions about the sequence of requests and predictions, we can find the optimal batch size. Also in the experiment we were able to get similar conclusions as before.

In the third part we discuss the OFTPL algorithm which considers the switching cost. We found that it still has a sublinear regret bound. When we consider perfect predictions, the bounds are better than the FTPL algorithm that uses dynamic learning rates without considering predictions. At the same time, if we face a higher switching cost, the algorithm with predictions will not necessarily perform better than the algorithm without prediction.

In the last part, we also found that the prediction accuracy also has a significant impact on our algorithm. When we choose perfect predictions, the OFTPL algorithm is able to achieve a smaller average cost than the FTPL algorithm in our case.

But when its accuracy rate drops, its average cost is gradually increasing. From our experimental results we can see that a high accuracy is required in case that requests has stable distribution to make the OFTPL algorithm perform better. But it also shows that in the example of considering the switching cost, when we are faced with requests with a stable distribution, we can also obtain a good performance without considering the predictions. However, in actual traces, it shows that even suboptimal predictions can yield satisfactory results.

In addition to the above conclusions, some new problems were also discovered when completing this work. For example, how the switching cost specifically affects the decision in the caching problem, and how to make the algorithm stable under more complex request situations. These represent possible future works.

Bibliography

[1] Gerhard Hasslinger, Juho Heikkinen, Konstantinos Ntougias, Frank Hasslinger, and Oliver Hohlfeld. Optimum caching versus lru and lfu: Comparison and combined limited look-ahead strategies. In 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), pages 1–6. IEEE, 2018.

[2] Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. Learning to cache with no regrets. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pages 235–243. IEEE, 2019.

[3] Shai Shalev-Shwartz et al. Online learning and online convex optimization. Foundations and Trends® in Machine Learning, 4(2):107–194, 2012.

[4] Naram Mhaisen, George Iosifidis, and Douglas Leith. Online caching with optimistic learning. In 2022 IFIP Networking Conference (IFIP Networking), pages 1–9. IEEE, 2022.

[5] Keyi Chen and Francesco Orabona. Generalized implicit follow-the-regularized-leader. arXiv preprint arXiv:2306.00201, 2023.

[6] Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Perturbation techniques in online learning and optimization. Perturbations, Optimization, and Statistics, 233, 2016.

[7] Peter J Denning. Virtual memory. ACM Computing Surveys (CSUR), 2(3):153–189, 1970.

[8] Tolga Bektas, Osman Oguz, and Iradj Ouveysi. Designing cost-effective content distribution networks. Computers & Operations Research, 34(8):2436–2449, 2007.

[9] Nitul Dutta, Shobhit K Patel, Osama S Faragallah, Mohammed Baz, and Ahmed Nabih Zaki Rashed. Caching scheme for information-centric networks with balanced content distribution. International Journal of Communication Systems, 35(7):e5104, 2022.

[10] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. No-regret caching via online mirror descent. In ICC 2021-IEEE International Conference on Communications, pages 1–6. IEEE, 2021.

[11] Samrat Mukhopadhyay and Abhishek Sinha. Online caching with optimal switching regret. In 2021 IEEE International Symposium on Information Theory (ISIT), pages 1546–1551. IEEE, 2021.

[12] Francescomaria Faticanti and Giovanni Neglia. Optimistic online caching for batched requests. In ICC 2023-IEEE International Conference on Communications. IEEE, 2023.

[13] Rupert Freeman, David Pennock, Chara Podimata, and Jennifer Wortman Vaughan. No-regret and incentive-compatible online learning. In International Conference on Machine Learning, pages 3270–3279. PMLR, 2020.

[14] Gijs Admiraal. Online caching through optimistic online mirror descent. 2022.

[15] Weiran Wang and Canyi Lu. Projection onto the capped simplex. arXiv preprint arXiv:1503.01002, 2015.

[16] Georgios S Paschos, Apostolos Destounis, and George Iosifidis. Learning to cooperate in d2d caching networks. In 2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), pages 1–5. IEEE, 2019.

[17] Yunwen Lei and Ding-Xuan Zhou. Convergence of online mirror descent. Applied and Computational Harmonic Analysis, 48(1):343–373, 2020.

[18] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. Foundations and Trends® in Machine Learning, 8(3-4):231–357, 2015.

[19] Yun Kuen Cheung and Georgios Piliouras. Online optimization in games via control theory: Connecting regret, passivity and poincaré recurrence. In International Conference on Machine Learning, pages 1855–1865. PMLR, 2021.

[20] Elad Hazan et al. Introduction to online convex optimization. Foundations and Trends® in Optimization, 2(3-4):157–325, 2016.

[21] Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and George Iosifidis. Optimistic no-regret algorithms for discrete caching. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 6(3):1–28, 2022.

[22] Mehryar Mohri and Scott Yang. Accelerating online convex optimization via adaptive prediction. In Artificial Intelligence and Statistics, pages 848–856. PMLR, 2016.

[23] Peter Auer, Nicolo Cesa-Bianchi, and Claudio Gentile. Adaptive and self-confident on-line learning algorithms. Journal of Computer and System Sciences, 64(1):48–75, 2002.

[24] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. Fundamental limits on the regret of online network-caching. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 4(2):1–31, 2020.

[25] Alon Cohen and Tamir Hazan. Following the perturbed leader for online structured learning. In International Conference on Machine Learning, pages 1034–1042. PMLR, 2015.

[26] Jacob Abernethy, Chansoo Lee, Abhinav Sinha, and Ambuj Tewari. Online linear optimization via smoothing. In Conference on Learning Theory, pages 807–823. PMLR, 2014.

[27] Debjit Paria and Abhishek Sinha. Leadcache: Regret-optimal caching in networks. Advances in Neural Information Processing Systems, 34:4435–4447, 2021.

[28] Yuanyuan Li, Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. Online caching networks with adversarial guarantees. ACM SIGMETRICS Performance Evaluation Review, 50(1):91–92, 2022.

[29] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time-aware cache algorithms. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 2(4):1–29, 2017.