1 Introduction

We have a fast memory, a cache, which can contain up to k different pages and we receive a sequence of requests to pages in an online manner. For instance,

$$P := ACFBG \tag{1}$$
$$\sigma = A, A, C, D, F, F, A, B \tag{2}$$

where σ is the sequence of page requests and $k = 4$. P is the cache content.

Whenever a page request is already included in the cache content, then it can be accessed at no cost, otherwise we suffer a *page fault.* Here one page must be evicted to make room for the request. This page to be evicted must be chosen in an online manner without knowledge of future requests. Therefore, we want to minimize the total number of page faults.

The online nature of caching and Metric Task Systems in general forces an algorithm to be suboptimal. This is why predictions are introduced. Nevertheless, the use of predictions does not guarantee an optimal behviour of an algorithm. Therefore, we evaluate it based on three elements:

- **Consistency**: performance with perfect predictions should be optimal;
- **Robustness**: performance with bad predictions should not be worse than what is achievable by known algorithms that do not utilize predictions;
- **Smoothness**: deterioration of performance with prediction errors should be smooth between consistency and robustness.

Given the computationally intensive nature of producing predictions, it is interesting to produce algorithms that are parsimonious in the number of predictions used. The following algorithms work with **action predictions** where each prediction is the state of an optimal algorithm at a given time step. We study how the algorithms would behave when we have a bound on the total number of predictions and when queries to a predictor can be accessed only after some defined time steps.

1.1 Results

The first contribution is an algorithm for caching that receives predictions emulating states of an optimal offline algorithm, Belady's algorithm.

Theorem 1.1. *Let f be an increasing convex function such that $f(0) = 0$ and $f(i) \leq 2^i - 1$ for each $i \geq 0$. There is an algorithm for caching requiring at most $f(logk)OPT$ predictions which achieves consistency 1, robustness $O(logk)$, and smoothness $O(f^{-1}(\rho/OPT))$, where ρ is the prediction error with respect to Belady and OPT is the number of page faults of Belady.*

Theorem 1.2. *Any $f^{-1}(\rho)$-smooth algorithm for caching with action predictions has to use a number of predictions bounded between $f(lnk)OPT \leq x \leq f(logk)OPT$.*

In terms of an MTS problem, the number of predictions cannot be bounded by a function of OPT, since the system can be scaled down to make OPT arbitrarily small. Given the small number of predictions that can be used, the algorithm proposed queries the predictor once in every time step a, thus making a total number of queries of $\frac{len(\sigma)}{a}$.

Theorem 1.3. *There is a deterministic algorithm for any MTS which receives a prediction only once per each a time steps and its cost is at most $O(a)\cdot(OFF+2\rho)$, where OFF denotes the cost of an arbitrary offline algorithm.*

This means that the algorithm for any MTS would have a competitive smoothness of

$$\frac{O(a) \cdot (OFF + 2\rho)}{OFF} = O(a) \cdot (1 + \frac{2\rho}{OFF}) \tag{3}$$

and with perfect predictions, hence $\rho = 0$, cosistency of $O(a)$. The same concept can be applied to caching algorithms thsu haveng predictor queries separated by a time steps, without loosing its stated properties in consistency, robustness and smoothness.

2 Preliminaries

2.1 Action Predictions for MTS

At each time t, the predictions tells us what state the optimal offline algorithm would be in. This will be denoted by p_t. On the other hand, o_t is the actual state of the offline algorithm at t. Thus, $\rho_t = d(p_t, o_t)$, where $d()$ is a distance function in the metric task system, defines the prediction error. The algorithm asks a specific number if pages that are present in its cache content and absent from that of the prediction. Of course, the bound is on the total number of queries as well as on the number of such indices.

2.2 Caching: Belady's algorithm, Marking and Lazy algorithms

The optiml offline algorithm used is Belady's, which, at every page fault, evicts the page that is going to be requested furthest in the future.
Properties of marking algorithms are used. Such algorithms split the input sequence into phases, which are maximal subsequences where at most k distinct pages are requested. Phases start at arbitrary moments, without a regular order. So, suppose $k = 3$ and $\sigma = [A, B, B, C, A, D, D, D, A, F, B, A, D, C]$. A possible partition in phases could be

$$\sigma_1 = [A, B, B, C] \tag{4}$$
$$\sigma_2 = [A, D, D, D, A, F] \tag{5}$$
$$\sigma_3 = [B, A, D] \tag{6}$$
$$\sigma_4 = [C] \tag{7}$$

Let O be the cache content at the beginning of a phase. For instance $O = DBA$, with $k = 3$. We start with the first phase.

- Page A is requested. This is in the cache. This is called an **arrival** and we mark the page by putting it in set M. so $M = [A]$.
- Page B is requested. Already in cache and also an arrival. $M = [A, B]$.
- Page B requested. Not an arrival. Already marked.
- Page C is requested. Not in cache. This is a page fault. Nevertheless, an arrival and $M = [A, B, C]$. C will be called a clean page since it is in M but not in O. A and B are old pages. The set of clean pages is $C = [C]$.

At the end of the phase, M has size k.
A marking algorithm is a type of algorithm that never evicts a marked page. At the end of the phase, the cache content is M.

In particular, when Belady evicts a certain p at time t, then that same page will not be requested in the marking phase containing t. Lastly, Belady is defined as a **lazy algorithm**, since it evicts only one page at a time and only at a page fault.