On Multi-threaded Metrical Task Systems[*]

Esteban Feuerstein[†] Steven S. Seiden[‡§] Alejandro Strejilevich de Loma[†]

January 11, 2006

Abstract

Traditionally, on-line problems have been studied under the assumption that there is a unique sequence of requests that must be served. This approach is common to most general models of on-line computation, such as Metrical Task Systems. However, there exist on-line problems in which the requests are organized in more than one independent thread. In this more general framework, at every moment the first unserved request of each thread is available. Therefore, apart from deciding *how* to serve a request, at each stage it is necessary to decide *which* request to serve among several possibilities.

In this paper we introduce Multi-threaded Metrical Task Systems, that is, the generalization of Metrical Task Systems to the case in which there are many threads of tasks. We study the problem from a competitive analysis point of view, proving lower and upper bounds on the competitiveness of on-line algorithms. We consider finite and infinite sequences of tasks, as well as deterministic and randomized algorithms. In this work we present the first steps towards a more general framework for on-line problems which is not restricted to a sequential flow of information.

Keywords: Competitive analysis, multi-tasking systems, on-line algorithms, paging

1 Introduction

Traditionally, on-line problems have been studied under the assumption that there is a unique sequence of requests that must be served. This means that at every moment there is one outstanding request that can be served, and each request becomes available when a decision is made on how to serve the preceding request in the sequence. This approach is common to most general theoretical frameworks for on-line problems, such as Metrical Task Systems [8] and Request-Answer Games [5].

Although the single-sequence approach is enough to model many on-line problems, it fails to encompass other interesting on-line situations of two different types. First, in certain problems there is a notion of *real-time* that passes while the requests are served, with the requests arriving over time. We will not deal with that kind of problems in this paper; for more information see, for example, [2, 15]. Second, there exist on-line problems in which the requests are not totally ordered but organized in more than one thread, as if they came from more than one source. These problems are called multi-threaded on-line problems. Problems of this type are usual, for instance, in multi-tasking environments, where several independent processes

[*]Research supported in part by UBACyT project "Modelos y Técnicas para Problemas de Optimización Combinatoria". Part of this work was done while Steven S. Seiden visited the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, supported by the Ministerio de Cultura y Educación project FOMEC 376.

[†]Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Pabellón I, Ciudad Universitaria, (1428) Capital Federal, Argentina; e-mail: {efeuerst,asdel}@dc.uba.ar.

[‡]Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA.

[§]Steve Seiden died in a tragic accident on June 11, 2002. The other authors sadly regret his loss.

may simultaneously present their requirements to the operating system. In this more general framework, at every moment the first unserved request of each thread is available. Therefore, apart from deciding *how* to serve a request, at each stage it is necessary to decide *which* request to serve among several possibilities.

The first multi-threaded on-line problem proposed in the literature was Multi-threaded Paging [11], which is the multi-threaded version of the traditional Paging problem [21, 24]. Multi-threaded Paging has been further studied in [13, 16, 22, 25]. More recently, multi-threaded versions of scheduling [12, 19], routing [1] and transportation problems [14, 23] have been proposed.

In this paper we introduce Multi-threaded Metrical Task Systems, that is, the generalization of Metrical Task Systems to the case in which there are many threads of tasks. We study the problem from a competitive analysis point of view, proving lower and upper bounds on the competitiveness of on-line algorithms. We consider finite and infinite sequences of tasks, as well as deterministic and randomized algorithms. With this work, we expect to make the first steps towards a more general framework for on-line problems which is not restricted to a sequential flow of information.

The remainder of this paper is organized as follows: In Section 2 we give basic definitions and notation. Section 3 is devoted to exploring related work that can be found in the literature. In Section 4 we present a parameterized algorithm for Multi-threaded Metrical Task Systems that achieves a competitive ratio that is essentially the product of the number of threads and the competitiveness of an algorithm for the corresponding single-sequence problem. In Section 5 we study an important class of Multi-threaded Metrical Task Systems, for which a lower bound on the competitive ratio is shown. In addition, we find concrete problems that have competitive ratios coincident with the lower bound that we give, and other problems that have competitive ratios coincident with the upper bound of Section 4; this means that our lower and upper bounds cannot be improved in general. In Section 6 we prove a lower bound for deterministic on-line algorithms that deal with infinite sequences of tasks; this lower bound is optimal up to constant factors. Finally, Section 7 is dedicated to presenting some remarks.

2 Definitions

A *task system* is given by a tuple (S, d, T, s_0). S is a finite set of *states*, where the number of states is $|S| = n$. To simplify the presentation, we consider that $S = \{1, 2, \ldots n\}$. d is a matrix giving the distances between states, i.e., for any two states i and j, $d(i,j) \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$ is the distance from i to j. We assume that the triangle inequality holds, and that $d(i,i) = 0$. The maximum and minimum distances are $d_{\max} = \max_{i \neq j} d(i,j)$ and $d_{\min} = \min_{i \neq j} d(i,j)$, respectively. T is a set of allowed tasks, where a *task* is a vector giving the costs of processing the task in the different states. More precisely, if t is a task, $t(i) \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$ specifies the cost of processing the task t while in the state i. The initial state is s_0.

An algorithm for a task system is presented with a sequence of tasks $\sigma = t^1, t^2, \ldots t^\ell$. The objective is to determine a state in which to process each task, minimizing the cost of moving and the cost of processing tasks. An algorithm produces a *schedule* $\pi = \pi^1, \pi^2, \ldots \pi^\ell$, where $\pi^i \in S$ is the state in which task t^i is processed. The cost of a schedule π on σ is the sum of the cost of moving from state to state, the *moving cost*, and the cost of processing tasks, the *stationary cost*:

$$C_\pi(\sigma) = \sum_{i=1}^{\ell} d(\pi^{i-1}, \pi^i) + \sum_{i=1}^{\ell} t^i(\pi^i) \ ,$$

where we define $\pi^0 = s_0$. The cost of an algorithm A on input σ, denoted $C_A(\sigma)$, is the cost of the schedule produced by the algorithm.

An on-line algorithm must decide in which state to process each task without knowledge of future tasks. Only after processing the current task, the next one is revealed. More formally, while producing a schedule π an on-line algorithm must determine each state π^i as a function only of $t^1, t^2, \dots t^i$. On the contrary, an off-line algorithm can decide each state based on the whole sequence of tasks.

A simple dynamic programming approach suffices to determine an optimal schedule for a sequence. The optimal off-line algorithm is an algorithm that produces always an optimal schedule; its cost for an input σ is

$$C_{OPT}(\sigma) = \min_{\pi} C_{\pi}(\sigma) \ .$$

The theoretical importance of task systems is that they can be used to model a wide variety of on-line problems. In other words, by appropriately choosing S, d, and T in the definition of task systems, it is possible to model specific on-line problems [8].

Competitive analysis is a type of worst case analysis where the performance of an algorithm is compared to that of the optimal off-line algorithm. This approach was initiated by Sleator and Tarjan [24]. The term *competitive analysis* originates in [18]. The measure of performance used in competitive analysis is the *competitive ratio*. In terms of task systems, it is defined as follows: An algorithm A for a task system is said to be *c-competitive* if and only if there exists a constant E such that for every task sequence σ

$$C_A(\sigma) - c \cdot C_{OPT}(\sigma) \leq E \ .$$

If the algorithm A is randomized then $C_A(\sigma)$ is a random variable and we use its expectation in the above definition. The *competitive ratio of algorithm A* is the infimum of the set of values c for which A is c-competitive. The *competitive ratio of task system M* is the infimum of the competitive ratio of A over all on-line algorithms A for M. We define $\det(M)$ and $\mathrm{ran}(M)$ to be the deterministic and randomized competitive ratios of task system M, respectively.

If in a task system M the distance matrix d is symmetric, we say that M is a *metrical* task system (MTS). In this situation we can assume that for any two states $i \neq j$, $0 < d(i,j) < +\infty$, since if $d(i,j) = 0$ the state i and the state j can be grouped together, and if $d(i,j) = +\infty$ the state i or the state j cannot be reached from the initial state. Since our performance measure is the competitive ratio, by scaling the distances and the tasks we can always assume that $d_{\min} = 1$.

As we mentioned before, there are on-line problems that can be better modeled with more than one sequence of tasks. With that goal, we introduce the notion of *multi-threaded task system*, which is a tuple (S, d, T, s_0, w). The parameters S, d, T and s_0 are defined as in a (single-threaded) task system, while the additional parameter w indicates the number of threads of tasks. An algorithm for a multi-threaded task system receives as input a w-tuple $\Sigma = (\sigma_1, \sigma_2, \dots \sigma_w)$ of sequences of tasks. Given its input, the algorithm produces a schedule for it. This schedule specifies not only the state in which to process each task, but the order in which the tasks are served. It may be helpfull to see a multi-threaded task system as a dynamic process in the following way: At every moment there is just one outstanding task per thread; the algorithm chooses one of those tasks and a state in which to process it; each time that the algorithm serves a task of a thread, the next task of that thread is presented to the algorithm. Notice that at each step distinct algorithms may have served distinct sets of tasks. However, the tasks of any particular sequence are served in the same order in which they are presented (although they can be interleaved with tasks from other sequences).

The cost of a schedule for an input tuple Σ is the sum of the moving cost and the stationary cost, just as in the single-threaded case. The cost of an algorithm is the cost of the schedule produced by the algorithm. Again we can distinguish between on-line and off-line algorithms: while algorithms of the latter type can decide the sequence of states and the ordering of the tasks based on the entire input, on-line algorithms must decide each step using only the information of the tasks seen so far.

If $M = (S, d, T, s_0)$ is a task system, we use $M(w)$ to denote the multi-threaded task system (S, d, T, s_0, w). We consider two models: finite and infinite. In the finite model the input Σ is a tuple of finite sequences of tasks, while in the infinite model each sequence contains an infinite number of tasks. In both models we use $C_A(\Sigma, \ell)$ to denote the cost incurred by algorithm A for serving ℓ tasks from the input tuple Σ; however, in the finite model we will usually use $C_A(\Sigma)$ instead of $C_A(\Sigma, |\Sigma|)$.

In the finite model algorithms are required to serve all tasks of all threads. At that point their performance is evaluated. An algorithm A is c-competitive in the finite model if and only if there exists a constant E such that for every input tuple Σ

$$C_A(\Sigma) - c \cdot C_{OPT}(\Sigma) \le E \ .$$

If A is a randomized algorithm we replace $C_A(\Sigma)$ for its expected value.

On the other hand, in the infinite model algorithms are required to serve a finite number ℓ of tasks; the value of ℓ is not known by on-line algorithms. An algorithm A is c-competitive in the infinite model if and only if there exists a constant E such that for every input tuple Σ and for every finite number ℓ of tasks

$$C_A(\Sigma, \ell) - c \cdot C_{OPT}(\Sigma, \ell) \le E \ ,$$

where $C_{OPT}(\Sigma, \ell)$ is the cost incurred by the optimal off-line algorithm for serving ℓ tasks from Σ. Again we replace $C_A(\Sigma, \ell)$ for its expectation if the algorithm A is randomized.

In both the finite and infinite models, we define the competitive ratio of an algorithm A and the competitive ratio of a multi-threaded task system $M(w)$ as in the single-threaded case. At any stage, a sequence σ_i whose last task has not been served yet is called *active*. The tuple formed by the jth task of each sequence is called the jth *row* of tasks.

A multi-threaded metrical task system (MT-MTS) is a multi-threaded task system in which the distance matrix d is symmetric. All of our results are for the metrical case.

Since the competitive ratio is a worst case measure, for the purposes of analysis we assume that the input sequences are generated by a malicious *adversary*, who forces on-line algorithms to perform as badly as possible. Thus, we use the terms *optimal off-line algorithm* and *adversary* interchangeably.

3 Related Work

Metrical Task Systems were introduced by Borodin, Linial, and Saks [8] as a generalization of several known single-threaded on-line problems. In that work, it was proved that if M is any MTS, then $\det(M) \le 2n - 1$; it was also proved that this upper bound is optimal. In contrast to the deterministic case where tight bounds have been attained, developing tight bounds for randomized algorithms has proven to be much less tractable. The best randomized lower bound for arbitrary metric spaces is

$$\Omega\left(\sqrt{\frac{\log n}{\log\log n}}\right) \ ,$$

which is due to Blum *et al.* [6]. The best randomized upper bound result is that of Bartal *et al.* [3] who proved that if M is any MTS, then $\operatorname{ran}(M) = O(\log^6 n)$. All the aforementioned results are for the case where all possible tasks are allowed. Burley and Irani [9] investigated the situation where T is given as part of the input.

So far only a small number of multi-threaded on-line problems have been analyzed. One of those problems is Multi-threaded Paging (MT-Paging), which is the multi-threaded version of the very well known Paging problem [21, 24]. MT-Paging was developed by Feuerstein [11] and Feuerstein and Strejilevich de Loma [16]. In those works, the authors considered finite and infinite sequences of requests, and they analyzed the problem both with and without imposing fairness restrictions, deriving deterministic lower and upper bounds. Further work on MT-Paging was done by Strejilevich de Loma [25], who considered some interesting particular cases of the fair version; by Feuerstein, Robak and Strejilevich de Loma [13], who improved some of the results for the finite model; and by Seiden [22], who gave randomized lower and upper bounds.

Another multi-threaded on-line problem that has been studied is the so-called *k-client problem*, due to Alborzi *et al.* [1]. The problem is the multi-threaded version of the 1-server problem [20]. In the k-client problem there is a metric space in which a server moves at constant speed to satisfy requests generated by k independent clients. Each request is satisfied when the server arrives to the location of the request, and then the corresponding client presents a new request in another place of the metric space. The problem was recently generalized by Seleson [23] and Feuerstein, Seleson and Strejilevich de Loma [14], who considered requests that consist of two points in the metric space, an origin and a destination; in this case the server must carry some object from the origin to the destination.

Feuerstein, Mydlarz and Stougie [12] have recently studied On-line Multi-threaded Scheduling, the problem of assigning a set of tasks presented by independent sequential clients to machines, in order to minimize some objective function such as the makespan or the latency.

Motivated by the problem of deciding which blocks of data to prefetch in a multi-tasking environment, Kimbrel [19] has analyzed the *sequence interleaving problem*. In this problem there are several sequences of positive and negative numbers. The goal is to interleave the sequences of numbers minimizing a cost function derived from the original prefetching problem.

Fiat and Karlin [17] have considered a problem related to MT-Paging, in which the input corresponds to a multi-pointer walk on an *access graph* [7]. Within that framework, the multiple threads of requests are merged in one input sequence, corresponding to an interleaved execution of the different threads. The way in which the sequences are interleaved in [17] is decided in an earlier stage of the process (and is the same for all algorithms). A similar approach was taken in [4, 10] for *application-controlled paging*. In this problem, a certain number of applications share a cache. Each application gives a sequence of requests to pages, and the algorithm must serve an interleaved request sequence.

4 A General Upper Bound

In this section, we will prove that the competitive ratio of any MT-MTS with w threads cannot be much worse than w times the competitive ratio of the corresponding single-threaded problem. In doing so, we will define an algorithm for MT-MTS that uses as a subroutine an algorithm for the single-threaded case.

Let M be any MTS and let A be any algorithm for M. Based on A we can define an algorithm for $M(w)$. We call the new algorithm Alternate-and-Restore (AR, for short), and it is described in Fig. 1. Algorithm AR receives as parameters the algorithm A and a positive real number g. Algorithm $AR(A, g)$ works in rounds; each round consists of applying algorithm A to each thread of the input tuple. The number of tasks of each thread served in each round depends on the parameter g. Before serving the first unserved task of any sequence during any round, $AR(A, g)$ moves to the state in which the algorithm was the last time that the sequence was served. In the finite model the algorithm must check whether the sequences are exhausted. We will now relate the competitiveness of $AR(A, g)$ with that of A.

Theorem 1 *Let M be any MTS. Let A be any deterministic or randomized c-competitive algorithm for M. Then for every $g > 0$ algorithm $AR(A, g)$ is $(wc + wd_{\max}/g)$-competitive for*

```
$r \leftarrow 1$
While there is at least one task to be served do
      % a new round starts
      $i \leftarrow 1$
      While $i \leq w$ do
            % a new stage starts
            Restore the state from the previous time that the $i$th sequence
            was served, or the initial state if the sequence was never served
            Apply algorithm $A$ to the $i$th sequence until the optimal off-line
            cost in the sequence is at least $gr$, or till the sequence is over
            $i \leftarrow i + 1$
      end While
      $r \leftarrow r + 1$
end While.
```

Figure 1: Algorithm Alternate-and-Restore(A, g).

$M(w)$ in both the finite and infinite models.

Proof: We will prove the result only for the deterministic case; for the randomized version the proof is almost the same, using the expectation of the cost.

Let ℓ be the number of tasks to be served (in the finite model, $\ell = |\Sigma|$). Suppose that after serving ℓ tasks $AR(A, g)$ completed m rounds and is currently in the $(m+1)$st round (so, $r = m + 1$ in Fig. 1). Note that, ignoring the restoring part, $AR(A, g)$ behaves exactly like algorithm A on each thread. Then the cost of $AR(A, g)$ for any thread is the cost of A for that thread plus the restoring cost. Since A is c-competitive its cost in each sequence is at most $cgm + O(1)$, while the restoring cost for each sequence is at most $d_{\max}m + O(1)$. Therefore the cost of $AR(A, g)$ is

$$C_{AR(A,g)}(\Sigma, \ell) \leq w(cgm + d_{\max}m) + O(1) = \left(wc + \frac{wd_{\max}}{g}\right) gm + O(1) \ .$$

Among the sequences in which $AR(A, g)$ completed its mth round (in the infinite model, all the sequences), there must be at least one sequence for which the adversary served at least the same number of tasks as $AR(A, g)$ in that sequence. By definition of $AR(A, g)$, the optimal off-line cost in that sequence is at least gm, and so

$$C_{AR(A,g)}(\Sigma, \ell) \leq \left(wc + \frac{wd_{\max}}{g}\right) C_{OPT}(\Sigma, \ell) + O(1) \ .$$

□

As we can see, an upper bound on the competitive ratio of any MT-MTS follows directly from the above theorem.

Corollary 2 *Let M be any MTS. For every $\epsilon > 0$ we have*

$$\det(M(w)) \leq w \cdot \det(M) + \epsilon \quad \textit{and} \quad \mathrm{ran}(M(w)) \leq w \cdot \mathrm{ran}(M) + \epsilon \ ,$$

in both the finite and infinite models.

Proof: By definition of $\det(\cdot)$, there exists a deterministic $[\det(M) + \epsilon/(2w)]$-competitive on-line algorithm A for M. Let $g = 2wd_{\max}/\epsilon$. Then, by Theorem 1, algorithm $AR(A, g)$ is $(w \det(M) + \epsilon)$-competitive for $M(w)$ in both models. For the randomized case, use a randomized $[\mathrm{ran}(M) + \epsilon/(2w)]$-competitive on-line algorithm. □

5 Forcing Tasks

In this section we will restrict our attention to an important class of MTS in which the set of allowed tasks can only contain *forcing tasks* [20]. A task t is a forcing task if and only if for any state i we have that $t(i)$ is either 0 or $+\infty$. This means that to process the task, every algorithm must change to a state in which the processing cost of the task is 0. Examples of MTS's with forcing tasks are the Paging problem [21, 24] and its generalization, the k-server problem [20].

We will see that, for any MTS with forcing tasks, the competitive ratio of the corresponding MT-MTS cannot be better than the competitive ratio with only one thread. After that, we will show that both the general upper bound of Section 4 and the lower bound of this section are achievable for some MTS's with forcing tasks. This could mean that, to improve any of those bounds, it would be necessary to make assumptions about the underlying metric space or the set of allowed tasks.

5.1 A Lower Bound

Theorem 3 *Let M be any MTS with forcing tasks. Then we have*

$$\det(M(w)) \geq \det(M) \quad \textit{and} \quad \operatorname{ran}(M(w)) \geq \operatorname{ran}(M) \ ,$$

in both the finite and infinite models.

Proof: Again we will present the proof only for the deterministic case; in a randomized setting, use the expectation of the cost. The idea of the proof is that an algorithm faced with w identical copies of a worst-case input sequence for the single-threaded problem, cannot behave better than its single-threaded counterpart.

Let $A(w)$ be any deterministic on-line algorithm for $M(w)$. We can use $A(w)$ to define an algorithm A for M as follows. Let σ be the input sequence of A. To process σ, algorithm A simulates the behavior of $A(w)$ on $\Sigma = (\sigma, \sigma, \dots \sigma)$. Each time that $A(w)$ moves to any state, so does A; each time that $A(w)$ serves the first task of the mth row of Σ, algorithm A serves the mth task of σ.

Note that A is well defined because when $A(w)$ has served wm tasks of Σ, algorithm $A(w)$ must have served the first task of the mth row. That is, if we call r_m the number of tasks served by $A(w)$ just after the algorithm has served the first task of the mth row, we have that $wm \geq r_m$. Since the cost is a non decreasing function of the number of served tasks, we have

$$C_{A(w)}(\Sigma, wm) \geq C_{A(w)}(\Sigma, r_m) \ .$$

By definition of A, the algorithm behaves almost in the same way as $A(w)$. The only difference is that when A has served m tasks of σ, it has served a subset of the r_m tasks of Σ served by $A(w)$, and hence

$$C_{A(w)}(\Sigma, r_m) \geq C_A(\sigma, m) \ .$$

Until now we did not use the fact that we are dealing with forcing tasks. We will use this to upper bound the cost of the adversary on Σ. To serve Σ, the adversary can follow an optimal off-line algorithm on σ, with the only distinction that each time a task of σ is served, the adversary serves a complete row of Σ. Since we have forcing tasks the cost of the adversary does not increase for serving those additional tasks, and so

$$C_{OPT}(\Sigma, wm) \leq C_{OPT}(\sigma, m) \ .$$

Based on the above discussion, we are ready to prove the claim. Let c be a constant such that $1 \leq c < \det(M)$, and let E be any constant. By definition of $\det(\cdot)$, there exists a sequence σ and a number of tasks ℓ (in the finite model, $\ell = |\sigma|$) such that

$$C_A(\sigma, \ell) - c \cdot C_{OPT}(\sigma, \ell) > E \ .$$

Consider the input tuple $\Sigma = (\sigma, \sigma, \ldots \sigma)$. In the infinite model, fix the number of tasks to serve in $w\ell$. In this situation we have

$$C_{A(w)}(\Sigma, w\ell) - c \cdot C_{OPT}(\Sigma, w\ell) \geq C_A(\sigma, \ell) - c \cdot C_{OPT}(\sigma, \ell) > E \ ,$$

and the result follows. □

5.2 An MT-MTS that Matches the Upper Bound

We will see now that there exist task systems for which the upper bound of Section 4 is optimal.

Theorem 4 *There exists a metrical task system M for which* $\det(M(w)) = w \det(M)$ *in the infinite model.*

Proof: This was proved in [16] for any MTS corresponding to the Paging problem with at least $w(k+1)$ distinct pages, where k is the size of the cache. □

Notice that in [22] it was proved that, up to constant factors, the analogous result is valid in a randomized setting.

5.3 An MT-MTS that Matches the Lower Bound

We will show now that the other extreme is possible, that there exist non-trivial task systems where the competitive ratio matches the lower bound of Theorem 3 for any number of sequences. In other words, for those task systems the competitive ratio is independent of the number of threads.

Theorem 5 *Let M be the MTS corresponding to the Paging problem with the restriction that the sequences of requests must be formed by at most $k+1$ distinct pages, where k is the size of the cache. Let A be any lazy [1] deterministic (randomized) c-competitive algorithm for M. Then there exists a deterministic (randomized) c-competitive algorithm for $M(w)$ in both the finite and infinite models.*

Proof: Once again we will give the proof only for the deterministic case. Based on A we will define an algorithm $A(w)$ for $M(w)$. Then we will prove that $A(w)$ is c-competitive.

Algorithm $A(w)$ starts by loading into the cache k different pages that appear in the input tuple Σ; this costs k to the algorithm. From that point on, $A(w)$ serves for free any request to a page that it has in its cache. This means that each time that $A(w)$ faults it is because the current request of each one of the active sequences is to its only missing page. To bring the missing page to its cache, $A(w)$ simulates the behavior of A on a request to that page.

Let ℓ be the number of requests to be served (in the finite model, $\ell = |\Sigma|$). Among the sequences that were active when $A(w)$ faulted the last time (in the infinite model, all the sequences) there must be at least one sequence for which the adversary served at least the same number of requests that $A(w)$ in that sequence. Let σ_j be such a sequence. Let $\overline{\sigma}_j$ be the subsequence of σ_j that contains only the requests in which $A(w)$ had a page fault after the initial loading of pages. Since σ_j was active when $A(w)$ faulted the last time, we can think that $A(w)$ served the ℓ requests just by loading the k different pages at the beginning, and then serving the requests in $\overline{\sigma}_j$; besides, the requests in $\overline{\sigma}_j$ were served by $A(w)$ using algorithm A, and therefore we have

$$C_{A(w)}(\Sigma, \ell) = k + C_A(\overline{\sigma}_j) \ .$$

[1] An algorithm for Paging is *lazy* if it only evicts a page on a page fault, and in that case evicts exactly one page.

Being A a c-competitive algorithm, there exists a constant E (independent of $\overline{\sigma}_j$) such that

$$C_A(\overline{\sigma}_j) \leq c \cdot C_{OPT}(\overline{\sigma}_j) + E \ .$$

By definition of σ_j, after serving ℓ requests the adversary has served all the requests in $\overline{\sigma}_j$, and then

$$C_{OPT}(\overline{\sigma}_j) \leq C_{OPT}(\Sigma, \ell) \ .$$

Putting the above three expressions together we obtain

$$C_{A(w)}(\Sigma, \ell) = C_A(\overline{\sigma}_j) + k \leq c \cdot C_{OPT}(\overline{\sigma}_j) + E + k \leq c \cdot C_{OPT}(\Sigma, \ell) + E + k \ ,$$

that is, $A(w)$ is c-competitive. □

Corollary 6 *There exists a metrical task system M for which* $\det(M(w)) = \det(M)$ *and* $\operatorname{ran}(M(w)) = \operatorname{ran}(M)$ *in both the finite and infinite models.*

Proof: Let M be the MTS corresponding to the Paging problem with the restriction that the sequences of requests must be formed by at most $k+1$ distinct pages, where k is the size of the cache. It is well known that there are lazy deterministic and randomized on-line algorithms for Paging that are optimal. Then the result follows by Theorem 3 and Theorem 5. □

6 An Additional Deterministic Lower Bound for the Infinite Model

In this section we will present a deterministic lower bound for the infinite model. The lower bound only assumes that the set T of allowed tasks contains some particular subset of tasks.

Theorem 7 *Let S be any set of states, with $n = |S|$. There exists a set of tasks T such that for every multi-threaded metrical task system $M(w) = (S, d, T, s_0, w)$, we have* $\det(M(w)) \geq wn$ *in the infinite model.*

Proof: Let A be any on-line algorithm for $M(w)$. The adversary picks $\epsilon > 0$ so that $1 \geq w\epsilon$. In addition, the adversary uses the following strategy: Whenever a task is revealed, that task charges ϵ to the current state of A and charges 0 to all other states.

We divide the schedule of A into phases. A phase ends, and a new one begins, whenever A changes state. Suppose that after serving ℓ tasks A completed m phases and is currently in the $(m+1)$st phase. Let p_i be the number of tasks served by A in the ith phase. Note that during a phase, all revealed tasks charge the current state of A. If A serves a task revealed in the current phase, it pays ϵ. Therefore the cost of A for the first phase is ϵp_1, because each available task charges the initial state and the algorithm has not moved. Consider the ith phase, with $i > 1$. During this phase, A can serve at most w tasks from previous phases. The algorithm moves before the phase ends and pays at least 1 for doing so (recall that $d_{\min} = 1$). Then the total cost for the phase is at least $1 + \epsilon(p_i - w)$. Since $\ell = \sum_{i=1}^{m+1} p_i$, summing over all phases we get

$$C_A(\Sigma, \ell) \geq \epsilon p_1 + \sum_{i=2}^{m+1} 1 + \epsilon(p_i - w) = (1 - w\epsilon)m + \epsilon\ell \geq \epsilon\ell \ .$$

Let q_i be the number of tasks served by A on thread σ_i. Notice that $\ell = \sum_{i=1}^{w} q_i$. Hence, there exists an i such that $q_i \leq \ell/w$. All tasks on this thread, starting with the $(q_i + 2)$nd task, charge 0 to all states. The adversary serves only tasks on this thread. Obviously, it pays nothing after the first $q_i + 1$ tasks. Furthermore, since each of the first $q_i + 1$ tasks charges exactly one state, there exists a state which is charged at most $(q_i + 1)/n$ times. Before serving any task the adversary moves to this state and pays at most $d_{\max} + \epsilon(q_i + 1)/n \leq d_{\max} + \epsilon/n + \epsilon\ell/(wn)$. Putting all these facts together, by simple calculations we obtain

$$\frac{C_A(\Sigma, \ell)}{C_{OPT}(\Sigma, \ell)} \geq \frac{\epsilon\ell}{d_{\max} + \epsilon/n + \epsilon\ell/(wn)} = \frac{wn}{wn(d_{\max} + \epsilon/n)/(\epsilon\ell) + 1} .$$

Clearly, this can be made arbitrarily close to wn by choosing sufficiently large ℓ. □

Recall that in [8] it was proved that if M is any MTS, then $\det(M) \leq 2n - 1$. Then, by Corollary 2, it follows that for every $\epsilon > 0$ we have $\det(M(w)) \leq w(2n - 1) + \epsilon = O(wn)$. This means that the result of Theorem 7 is tight up to constant factors.

7 Remarks

In this work, we have taken the first steps towards formulating a general model for multi-threaded on-line problems. Specifically, we have introduced Multi-threaded Metrical Task Systems, a natural generalization of the task system general model.

A central issue is the dependence of the competitive ratio on the number of threads w. We have shown that for metrical task systems, the competitive ratio is $O(wn)$. Further, we have exhibited metrical task systems where this is the best possible, and other metrical task systems where the competitive ratio is constant in w. An interesting open question is whether there are natural task systems which are inbetween, i.e., metrical task systems for which the competitive ratio grows with $f(w)$, where $f = o(w)$, and $f = \omega(1)$.

Another important issue is as follows: Given a metrical task system M, what is the natural multi-threaded version of M? The first impulse is to use $M(w)$. However, further consideration leads us to conclude that this is not entirely satisfactory. For instance, the most useful model of Multi-threaded Paging might be that where each thread requests its own set of pages. Given the wide range of problems which can be modeled as metrical task systems, we are not certain that there is a definitive answer to this question.

References

[1] Houman Alborzi, Eric Torng, Patchrawat Uthaisombut, and Stephen Wagner. The k-client problem. *J. Algorithms*, 41(2):115–173, 2001.

[2] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

[3] Y. Bartal, A. Blum, C. Burch, and A. Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proc. 29th ACM Symposium on Theory of Computing*, pages 711–719, 1997.

[4] Rakesh D. Barve, Edward F. Grove, and Jeffrey Scott Vitter. Application-controlled paging for a shared cache. *SIAM J. Comput.*, 29(4):1290–1303, 2000.

[5] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[6] Avrim Blum, Howard Karloff, Yuval Rabani, and Michael Saks. A decomposition theorem for task systems and bounds for randomized server problems. *SIAM J. Comput.*, 30(5):1624–1661, 2001.

[7] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, April 1995.

[8] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the Association for Computing Machinery*, 39(4):745–763, October 1992.

[9] W. Burley and S. Irani. On algorithm design for metrical task systems. *Algorithmica*, 18(4):461–485, August 1997.

[10] P. Cao, E.W. Felten, and K. Li. Application-controlled file caching policies. In *Proc. Summer USENIX Conference*, 1994.

[11] Esteban Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging*. PhD thesis, Università degli Studi di Roma "La Sapienza", 1995.

[12] Esteban Feuerstein, Marcelo Mydlarz, and Leen Stougie. On-line multi-threaded scheduling. *J. Scheduling*, 6(2):167–181, 2003.

[13] Esteban Feuerstein, Darío G. Robak, and Alejandro Strejilevich de Loma, 2000. Manuscript.

[14] Esteban Feuerstein, Mariela Seleson, and Alejandro Strejilevich de Loma, 2000. Manuscript.

[15] Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

[16] Esteban Feuerstein and Alejandro Strejilevich de Loma. On-line multi-threaded paging. *Algorithmica*, 32(1):36–60, January 2002.

[17] Amos Fiat and Anna R. Karlin. Randomized and multipointer paging with locality of reference. In *Proc. Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 626–634, Las Vegas, Nevada, 29 May–1 June 1995.

[18] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[19] Tracy Kimbrel. Interleaved prefetching. *Algorithmica*, 32(1):107–122, 2002.

[20] Mark S. Manasse, L.A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[21] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[22] Steven S. Seiden. Randomized online multi-threaded paging. *Nordic Journal of Computing*, 6(2):148–161, 1999.

[23] Mariela Seleson. On-line multi-threaded dial-a-ride. Master's thesis, Universidad de Buenos Aires, Departamento de Computación, July 1999.

[24] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of ACM*, 28:202–208, 1985.

[25] Alejandro Strejilevich de Loma. New results on fair multi-threaded paging. *Electronic Journal of SADIO*, 1(1):21–36, May 1998. `http://www.sadio.org.ar`.