Randomized Algorithms for Metrical Task Systems

Sandy Irani* Steve Seiden†
Department of Information and Computer Science
University of California
Irvine, CA 92717

December 17, 1997

Abstract

Borodin, Linial, and Saks introduce a general model for online systems in [BLS92] called *task systems* and show a deterministic algorithm which achieves a competitive ratio of $2n-1$ for any metrical task system with n states. We present a randomized algorithm which achieves a competitive ratio of $e/(e-1)n-1/(e-1) \approx 1.5820n-0.5820$ for this same problem. For the uniform metric space, Borodin, Linial, and Saks present an algorithm which achieves a competitive ratio of $2H_n$, and they show a lower bound of H_n for any randomized algorithm. We improve their upper bound for the uniform metric space by showing a randomized algorithm which is $\big(H_n + O(\sqrt{\log n})\big)$-competitive.

1 Introduction

In computer systems, it is often necessary to solve problems with incomplete information. The input evolves with time, and incremental computational decisions must be made based on only part of the input. A typical situation is where a sequence of tasks must be performed. How tasks are performed affects the cost of future tasks. Examples include managing a two level store of memory, performing a sequence of operations on a dynamic data structure, and maintaining data in a multiprocessing environment [KMMO94, MMS90, ST85b, Wes92]. An algorithm that decides how to perform a task based only on past requests with no knowledge of the future is said to be an *online algorithm*. In contrast,

*irani@ics.uci.edu, Research supported in part by NSF grant number CCR-9309456.
†sseiden@ics.uci.edu

we refer to an algorithm which has complete information about the tasks to be performed before it makes any decisions as an *offline algorithm*.

Borodin, Linial and Saks introduced *task systems* in [BLS92] as a way to model many particular online problems. In the model, states are used to represent the set of possible algorithm configurations. The cost of moving from one particular configuration to another is specified by a state transition cost matrix. The results for general metrical task systems often yield very weak results for particular problems because any special regularity a problem may have is lost in the generality of the definition. Nonetheless, metrical task systems are an important part of the existing general theory of online algorithms. In addition, work on metrical task systems have yielded some very important techniques and ideas.

A task system is defined as a pair (S, d) where $S = \{s_1, \ldots, s_n\}$ is a set of n states and $d = (d_{s,t})$, the *distance matrix*, is an $n \times n$ non-negative matrix. The distance from s to t is $d_{s,t}$. We assume that $d_{s,s} = 0$ for all s and d obeys the triangle inequality. We say that a task system is *metrical* if the distance matrix is also symmetric. An input to the system consists of a sequence of *tasks*, $\sigma = T^1, T^2, \ldots, T^\ell$. A task is a vector with n non-negative entries, where $T^i(s)$ is the cost of processing task i in state s. We say that task T *charges* state s if and only if $T(s) \neq 0$. The algorithm begins in state s_1. The objective is to determine a state in which to process each task, balancing the cost of moving with the cost of processing tasks. An algorithm produces a schedule π, a function from $\{0, 1, \ldots, \ell\}$ to S. We define $\pi(0) = s_1$. For $i > 0$, $\pi(i)$ is the state in which task i is processed. If the algorithm is online, then $\pi(i)$ is a function only of $T^1, \ldots, T^i$. 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*):

$$\mathrm{cost}(\pi, \sigma) = \sum_{i=1}^{\ell} d_{\pi(i-1),\pi(i)} + \sum_{i=1}^{\ell} T^i(\pi(i)).$$

We denote by $A(\sigma)$ the schedule produced by algorithm A on input σ. The *cost of algorithm A on σ* denoted $\mathrm{cost}_A(\sigma)$ is $\mathrm{cost}(A(\sigma), \sigma)$. The cost of the optimal offline algorithm for the sequence π is

$$\mathrm{cost}(\sigma) = \min_{\pi} \mathrm{cost}(\pi, \sigma).$$

A simple dynamic programming approach suffices to determine the optimal offline schedule for a sequence.

We evaluate an online algorithm by comparing its performance to that of the optimal offline algorithm. An online algorithm A is said to be c-competitive if there is a constant d such that for all σ,

$$\mathrm{cost}_A(\sigma) \leq c \cdot \mathrm{cost}(\sigma) + d.$$

If the algorithm A is randomized, then its cost on a given sequence $c_A(\sigma)$ is a random variable. We compare the expectation of this cost to the cost of the optimal algorithm on σ: A is said to be c-competitive if there is a constant d such that for all σ,

$$\mathrm{E}[\mathrm{cost}_A(\sigma)] \leq c \cdot \mathrm{cost}(\sigma) + d.$$

The *competitive ratio* of A is the infimum over all c such that A is c-competitive. This approach to analyzing online problems, called *competitive analysis*, was initiated by Sleator and Tarjan, who used it to analyze the List Update problem [ST85a]. The term *competitive analysis* originated in [KMMO94]. The goal for a given task system (S, d) is to determine the best competitive ratio achievable on that task system and the algorithm that achieves it. Since the competitive ratio is a worst case measure, for the purposes of analysis we assume that the task sequence is generated by a malicious *adversary*, who forces the algorithm to perform as badly as possible. Thus, we use the terms *optimal offline cost* and *adversary's cost* interchangeably. For the purposes of analyzing randomized algorithms, there are several types of adversaries [BDBK$^+$94]. We utilize an *oblivious* adversary; an adversary that does not know the random choices of the algorithm before determining the input sequence.

In the case of deterministic algorithms, Borodin, Linial, and Saks show in [BLS92] that for every metrical task system (S, d) the competitive ratio is exactly $2n - 1$. 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. Before this paper, there was no known randomized algorithm which was provably better than the optimal deterministic algorithm for general metric spaces. We show the first such algorithm, the Threshold Work Function algorithm, and show that this algorithm is

$$\left(\frac{e}{e-1}\right) n - \frac{1}{e-1}$$

competitive against an oblivious adversary. In contrast, the best lower bound for arbitrary metric spaces is

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

which is due to Blum, Karloff, Rabani and Saks [BKRS92]. Although they prove the lower bound for the k-server problem, their result applies to metrical task systems as well.

Since the problem of designing randomized algorithms for general metric spaces seems to be very difficult, researchers have turned to the more approachable problem of addressing specific metric spaces. Borodin *et al.* show a lower bound of H_n and an upper bound of $2H_n$ for the uniform distance matrix on n

states ($d_{s,t} = 1$ for all $s \neq t$). We call such tasks systems *uniform task systems*. In an earlier version of this paper, we present an upper bound of

$$\frac{H_n}{\ln 2} + 1$$

on the competitive ratio for uniform task systems [IS95]. In this work, we present a randomized algorithm for uniform task systems, which we call EXPONENTIAL, that achieves a competitive ratio of

$$H_n + O\left(\sqrt{\log n}\right)$$

against an oblivious adversary. Disregarding lower order terms, this bound is optimal. The only other metric space for which a randomized algorithm is known that out-performs the optimal deterministic one is a result due to Chrobak and Noga [CN] who show that the randomized competitive ratio for two states is exactly 2.

In Section 2 we describe EXPONENTIAL. In Section 3 we show an upper bound for the algorithm. In Section 4 we describe the THRESHOLD WORK FUNCTION algorithm. In Section 5 we present an upper bound on the competitiveness of this algorithm. In Section 6 we present open problems. Finally, in the appendix, we show a lower bound for the THRESHOLD WORK FUNCTION algorithm.

2 The EXPONENTIAL Algorithm

For convenience, we adopt a continuous time model also used in [BLS92] where state transitions can be made in the middle of the discrete time intervals. A *continuous time schedule* for ℓ tasks is a function from the continuous interval $[0, \ell)$ to S such that for each state s, $\pi^{-1}(s)$ is a finite disjoint union of half open intervals $[t, t')$. In addition we require that $\pi(0) = s_1$. There are a finite number of transition times $t_1 < t_2 < \cdots < t_k$. Denote the state to which the algorithm moves at time t_i by $x_i = \pi(t_i)$. We define $t_0 = 0$. The cost for the schedule is then

$$\text{cost}(\pi, \sigma) = \sum_{i=1}^{k} d_{x_{i-1}, x_i} + \sum_{i=1}^{\ell} \int_{i-1}^{i} T^i\left(\pi(t)\right) dt.$$

Allowing an algorithm the freedom to move at any time in a continuous time interval supplies no additional power. Borodin *et al.* prove the following lemma:

Lemma 2.1 *For any online continuous time algorithm A, there is an online discrete time algorithm A' that performs at least as well on any sequence of tasks.*

In order to describe EXPONENTIAL, we need one more definition: the *work function* for a sequence of tasks. The work function w_i maps states to non-negative real numbers. Intuitively, $w_i(s)$ is the minimum cost required to process the first i tasks and end up in state s. The idea of work functions was introduced in [BLS92]. The actual term *work function* was coined in [CL91]. For all s, we have $w_0(s) = d_{s,s_1}$. The work function for $i > 0$ is calculated inductively as follows:

$$w_{i+1}(u) = \min_v \{w_i(v) + T^{i+1}(v) + d_{u,v}\}.$$

When we omit the subscript i we are referring to the work function at the current point in the task sequence. Note that $\min_s w(s)$ is the optimal offline cost. Also note that $w(u) \leq w(v) + d_{u,v}$ for any two states u and v. Since in this case the distance matrix is uniform, no work function value is more than one plus the minimum work function value.

We extend the definition of the work function to the continuous time interval. Let i be an integer in $\{0, \ldots, \ell\}$ and λ be a real number such that $0 \leq \lambda < 1$. The work function at time $i + \lambda$ is

$$w_{i+\lambda}(u) = \min_v \{w_i(v) + \lambda T^{i+1}(v) + d_{u,v}\}.$$

Note that for any positive integer i, $w_i(s)$ is the same under the extended definition. Also note that $w_{i+\lambda}(s)$ is continuous with λ for $\lambda \in [i, i+1)$ for all states s.

We are now ready to describe the algorithm, which has a positive integer parameter m. The algorithm breaks time into non-overlapping periods called *phases*. Phase 0 begins at the start of the request sequence. Phase i ends, and phase $i+1$ begins, when the minimum work function value over all states reaches $i+1$. During phase i we define the *normalized* work function $\bar{w}$ by

$$\bar{w}_{i+\lambda}(s) = w_{i+\lambda}(s) - i.$$

Note that at the beginning of each phase the minimum value is zero (and all other values are ≤ 1). We define the *rank* of a state s to be

$$\operatorname{rank}(s) = \begin{cases} 0 & \text{if } \bar{w}(s) \geq 1, \\ \lfloor m\,(1 - \bar{w}(s)) \rfloor & \text{otherwise.} \end{cases}$$

The *weight* of a state s is defined to be

$$q(s) = \begin{cases} 0 & \text{if } \bar{w}(s) \geq 1, \\ 2^{\operatorname{rank}(s)-1} & \text{otherwise.} \end{cases}$$

Because the algorithm uses this exponential weight function, we call it EXPONENTIAL. Define

$$Q = \sum_s q(s).$$

We define the algorithm using a distributional approach. During any phase the probability that EXPONENTIAL is in a state s is

$$p(s) = \frac{q(s)}{Q}.$$

Note that the distribution changes only when the rank of some state changes. Furthermore, in order to maintain the distribution, the algorithm need only move when the distribution changes. If several states change rank simultaneously, we define the behavior of EXPONENTIAL as follows: the algorithm picks some arbitrary ordering of these states, and then changes its distribution as if the rank changes occurred in that order.

In order to describe precisely how the algorithm maintains the distribution, consider a single step in which only the rank of a single state u changes. Let p and p' denote the probability distribution before u changes rank and after u changes rank, respectively. Let $\Delta p(v) = p'(v) - p(v)$ for all $v \in S$. Note that $\Delta p(u) \leq 0$, and that $\Delta p(v) \geq 0$ for $v \neq u$. The probability that the algorithm moves to state v given that it is in u before the step is $\Delta p(v)/p(u)$. The probability that the algorithm stays in u given that it is in u is $(p(u) + \Delta p(u))/p(u)$. Thus the distribution can be maintained by following the rule: If the algorithm is in a state u which changes rank, pick a random state $v \in S$ with the distribution

$$\Pr[v] = \begin{cases} \frac{\Delta p(v)}{p(u)} & \text{if } v \neq u \\ \frac{p(u)+\Delta p(u)}{p(u)} & \text{otherwise.} \end{cases}$$

and move to v.

3 Analysis

For the analysis, we make the following restrictions on the adversary:

1. The adversary charges one state at a time.
2. The adversary charges states in *units* of $1/m$.

Assumption 2 is justified as follows: Note first that this assumption has no effect on the moving cost, since the algorithm's distribution changes only when a state's work function reaches a multiple of $1/m$. Secondly, note that during any phase, the probability that the algorithm is in a particular state decreases only when that state is charged. Suppose that the adversary makes a charge less than $1/m$ to a state s, charges some other states, and then completes the $1/m$ charge to s. The probability that the algorithm is in s at any time before the charge is completed is upper bounded by the probability the algorithm is in state s at the time of the charge's completion. This means that the stationary cost to the algorithm is at least as large if the $1/m$ charge was made all at once after the other states have been charged. Thus, for every sequence for which Assumption 2 does not hold, there is a sequence for which Assumption 2 does hold such the optimal offline algorithm incurs the same cost and the cost of the algorithm is at least as large.

Assumption 1 is justified as follows: The algorithm changes distribution only if some state changes rank. Suppose that two or more states change rank simultaneously. The algorithm changes distribution as if they change rank in some arbitrary order. So the cost is the same as if the states are charged in that order.

We analyze the cost of a single phase. First we compute the algorithm's moving cost. Consider a charge to some state s which the algorithm is in with positive probability. Before the charge

$$\bar{w}(s) = 1 - \frac{i}{m},$$

for some $i \in \{0, \ldots, m-1\}$, and after the charge $\bar{w}(s)$ increases by $1/m$. Let $q'(s)$ and Q' denote the values of $q(s)$ and Q after the charge. Since s is the only state whose rank changes, we know that $q(s) - q'(s) = Q - Q'$. Furthermore, since for any state t, $\bar{w}(t)$ only changes by increments of $1/m$, $q(t)$ (and hence Q) is always an integer. The probability that the algorithm moves from s is simply the change in probability that the algorithm is in state s:

$$\begin{aligned}
\frac{q(s)}{Q} - \frac{q'(s)}{Q'} &= \sum_{i=0}^{Q-Q'-1} \left[\frac{q(s)-i}{Q-i} - \frac{q(s)-(i+1)}{Q-(i+1)} \right] \\
&\leq \sum_{i=0}^{Q-Q'-1} \frac{1}{Q-i} = \sum_{i=0}^{Q} \frac{1}{i} - \sum_{i=0}^{Q'} \frac{1}{i}.
\end{aligned}$$

In addition, the algorithm incurs a cost of at most 1 at the beginning of each phase, when it changes from the distribution of the previous phase to the distribution of the current phase.

Notice that when a state is charged $1/m$, the change in the weight of the state is at least half of the initial weight of the state. Thus, the stationary cost that the algorithm incurs is

$$\begin{aligned}
\frac{1}{m}\frac{q(s)}{Q} &\leq \frac{2(q(s)-q'(s))}{mQ} = \frac{2(Q-Q')}{mQ} \\
&\leq \frac{2}{m} \sum_{i=0}^{Q-Q'-1} \frac{1}{Q-i} \\
&= \frac{2}{m} \left[\sum_{i=0}^{Q} \frac{1}{i} - \sum_{i=0}^{Q'} \frac{1}{i} \right].
\end{aligned}$$

Since Q is initially at most $n2^{m-1}$, when we sum the moving and stationary costs over all charges, we find the total cost for the phase is at most

$$\begin{aligned}
\left(1+\tfrac{2}{m}\right)\sum_{i=1}^{n2^{m-1}} \frac{1}{i} + 1 &= \left(1+\tfrac{2}{m}\right) H_{n2^{m-1}} + 1 \\
&\leq \left(1+\tfrac{2}{m}\right)\left(\ln(n2^{m-1}) + 1\right) + 1 \\
&= \left(1+\tfrac{2}{m}\right)\left(\ln n + (m-1)\ln 2 + 1\right) + 1 \\
&\leq \left(1+\tfrac{2}{m}\right)\left(H_n + m\ln 2 + 1\right) + 1 \\
&= H_n + \frac{2H_n+2}{m} + m\ln 2 + 2\ln 2 + 2.
\end{aligned}$$

We use the fact that $\ln n \leq H_n \leq \ln n + 1$ for $n > 1$ in the second and forth steps above. Since the adversarial cost for the phase is 1, the above bound is also the competitive ratio. To find the best choice of m, we consider

$$\frac{d}{dm}\left(H_n + \frac{2H_n+2}{m} + m\ln 2 + 2\ln 2 + 2\right) = -\frac{2H_n+2}{m^2} + \ln 2,$$

which is zero at

$$m = \sqrt{2(H_n+1)/\ln 2}.$$

Note that the second derivative is positive at this value of m, and so this is the value of m we chose. So the competitive ratio of Exponential is

$$\begin{aligned}
& H_n + \frac{2H_n+2}{\sqrt{2(H_n+1)/\ln 2}} + \ln 2\sqrt{2(H_n+1)/\ln 2} + 2\ln 2 + 2 \\
= \; & H_n + 2\sqrt{2\ln 2(H_n+1)} + 2\ln 2 + 2 \\
= \; & H_n + O\left(\sqrt{\log n}\right).
\end{aligned}$$

However note that this is not better than the previous bound of Irani and Seiden [IS95] until $n \approx 10^{30}$.

4 The Threshold Work Function Algorithm

To design a randomized algorithm which is competitive for any metric space, we begin with a deterministic algorithm which is known to be competitive, and 'add in' randomization. The algorithm which we randomize is a variant of the Work Function algorithm, introduced by Borodin, Linial and Saks [BLS92]. The basic idea of the (continuous time) Work Function algorithm is as follows: When the work function of the current state s reaches a value where

$$w(s) = w(t) + d_{s,t}$$

for some other state t, the algorithm moves to t. A simple variant of this is as follows: For some real number $\alpha \in (0,1]$, when the work function of the current state s reaches a value where

$$w(s) = w(t) + \alpha \cdot d_{s,t}$$

for some other state t, the algorithm moves to the state t. We call this generalized version the Alpha Work Function algorithm, after an algorithm of the same name for the k-server and layered graph traversal problems [CL91, Bur93]. Borodin, Linial and Saks show that the Work Function algorithm is $(2n-1)$-competitive [BLS92]. Further, they show that the Work Function algorithm is optimal; any deterministic algorithm for metrical task systems has a competitive ratio of at least $2n-1$. In the appendix, we show that the Alpha Work Function algorithm is $(\frac{\alpha+1}{\alpha}n - \frac{1}{\alpha})$-competitive.

By using randomization, we achieve a lower competitive ratio. We investigate the algorithm which chooses α randomly. The algorithm has a parameter m which is a positive integer. The algorithm chooses α from $(0,1]$ as follows: The probability that $\alpha \in \left(\frac{i-1}{m}, \frac{i}{m}\right]$ is p_i. If $\alpha \in \left(\frac{i-1}{m}, \frac{i}{m}\right]$ then it is uniformly chosen from this interval. We derive $p_1, \ldots, p_m$ in Section 5. We refer to the randomly chosen α as the *threshold*. We call this randomized algorithm the Threshold Work Function algorithm.

We again adopt a continuous time model. A formal description of the Threshold Work Function algorithm appears in Figure 1.

5 Analysis

We require a few definitions. Let

$$P_{i,j} = \sum_{k=i}^{j} p_k.$$

We define

$$\begin{aligned} \phi_s(t) &= \frac{w(s) - w(t)}{d_{s,t}}, \qquad \text{if } t \neq s, \\ \phi_s(s) &= 0. \end{aligned}$$

We define $\Phi_s = \max_t \phi_s(t)$. Let $T_s = \{t \in S | \phi_s(t) = \alpha\}$. T_s describes the states to which the algorithm might move to if $\Phi_s = \alpha$ and the algorithm is in state s. If $|T_s| = 1$ the algorithm only has one choice of states to move to, and we let $\eta(s)$ be the single member of T_s. Otherwise, we break ties as the algorithm does, $t = \eta(s)$ is the least indexed state with $\Phi_t < \alpha$. That this is always possible is shown in the following lemma.

Lemma 5.1 *There exists a $t \in T_s$ such that $\Phi_t < \alpha$.*

Algorithm THRESHOLD WORK FUNCTION

1. $a \leftarrow 1, \ldots, m$ randomly with distribution $\Pr[a = i] = p_i$.
2. $b \leftarrow (0, \frac{1}{m}]$ randomly and uniformly.
3. $\alpha \leftarrow \frac{a-1}{m} + b$.
4. $u \leftarrow s_1$.
5. Loop until end of task sequence:
   (a) While $w(u) < w(s) + \alpha \cdot d_{u,s}$ for all s
      i. Stay in u.
   (b) When there is a state s such that $w(u) = w(s) + \alpha \cdot d_{u,s}$,
      i. $u \leftarrow s$. If several states s exist such that $w(u) = w(s) + \alpha \cdot d_{u,s}$, pick the least indexed state s_i such that $w(s_i) < w(s') + \alpha \cdot d_{s_i,s'}$ for all s'.
      ii. Move to s.

Figure 1: A description of the THRESHOLD WORK FUNCTION algorithm.

Proof Let t_1 be some member of T_s. By definition $\phi_s(t_1) = \alpha$. If $\Phi_{t_1} < \alpha$ we are done. We show that $\Phi_{t_1} > \alpha$ is not possible. Suppose this were true. This would imply the existence of a state t_2 such that

$$w(t_1) = w(t_2) + \beta \cdot d_{t_1,t_2},$$

with $\beta > \alpha$. We have

$$\begin{aligned} w(s) &= w(t_1) + \alpha \cdot d_{s,t_1} \\ &= w(t_2) + \beta \cdot d_{t_1,t_2} + \alpha \cdot d_{s,t_1} \\ &> w(t_2) + \alpha \cdot d_{t_1,t_2} + \alpha \cdot d_{s,t_1} \\ &\geq w(t_2) + \alpha \cdot d_{s,t_2}. \end{aligned}$$

And so $\phi_s(t_2) > \alpha$, a contradiction. If $\Phi_{t_1} = \alpha$ then there exists a $t_2 \neq t_1$ with

$$w(t_1) = w(t_2) + \alpha \cdot d_{t_1,t_2}.$$

We show that $\phi_s(t_2) = \alpha$ and therefore $t_2 \in T_s$. Consider

$$\begin{aligned} w(s) - w(t_2) &= w(s) - w(t_1) + w(t_1) - w(t_2) \\ &= \alpha \cdot d_{s,t_1} + \alpha \cdot d_{t_1,t_2} \\ &\geq \alpha \cdot d_{s,t_2}. \end{aligned}$$

And so $\phi_s(t_2) \geq \alpha$. But $\phi_s(t_2) > \alpha$ is a contradiction, and so $t_2 \in T_s$.

This gives rise to a sequence of states $t_1, t_2, \ldots$. For each state t_i which is not the last state in the sequence, we know that $\Phi_{t_i} = \alpha$. t_i implies the existence of a t_{i+1} such that either either $\Phi_{t_{i+1}} < \alpha$ and the sequence ends, or $\Phi_{t_{i+1}} = \alpha$ and the sequence continues. Furthermore, we know that there are no repetitions in the sequence since

$$w(t_i) = w(t_{i+1}) + \alpha \cdot d_{t_i, t_{i+1}} > w(t_{i+1}).$$

Since there are only a finite number of states, we cannot have a infinite chain $t_1, t_2, t_3, \ldots$. Eventually we reach a t_k with $\Phi_{t_k} < \alpha$. ∎

Corollary 5.1 *The algorithm is never in a state s with $\Phi_s \geq \alpha$.*

Note that for every state s, $0 \leq \Phi_s \leq 1$. The lower bound $0 \leq \Phi_s$ is due to the fact that $\phi_s(s) = 0$, and so $\max_{t \in S} \phi_s(t)$ is at least 0. The upper bound is due to the fact that the work function of a state s is no more that $d_{s,t}$ greater than the work function of any other state t. Also note that since all work functions change continuously with time, Φ_s also changes continuously, since when $\eta(s)$ changes from t to t' we have that $\phi_s(t) = \phi_s(t')$.

We define the *useful work* of a task T^i to be the amount by which T^i increases work functions. Formally, the useful work u_i of T^i is

$$u_i = \sum_{s \in S} w_i(s) - w_{i-1}(s).$$

Lemma 5.2 *For a sequence of tasks σ let u_i be the useful work of task T^i and let $U_\ell = \sum_{i=1}^{\ell} u_i$. Then the optimal offline cost to serve $T^1, \ldots, T^\ell$ is at least*

$$\frac{U_\ell}{n} - d_{\max},$$

where $d_{\max} = \max_{s,t \in S} d_{s,t}$.

Proof The total increase of all work functions is the total useful work. There is at least one state s whose work function value is at least $\sum u_i / n$. For all t we have $w_\ell(t) \geq w_\ell(s) - d_{s,t}$. And so, for all t we have $w_\ell(t) \geq w_\ell(s) - d_{\max}$. Note that the optimal offline cost is $\min_{s \in S} w_\ell(s)$. ∎

A *period* with respect to a state s is a continuous time interval during which:

1. $\Phi_s < 1$.
2. $\eta(s)$ remains fixed.

3. Φ_s is either strictly increasing (in which case the period is an *increasing period*), or monotonically non-increasing (in which case the period is an *non-increasing period*), but not both.

A *sub-period* with respect to s is a continuous sub-interval of a period with respect to s.

A *maximal period* with respect to s is a period which is not a sub-period of any other period with respect to s.

We analyze the competitiveness of the algorithm against an oblivious adversary. We bound the expected cost by summing over all states s, the expected stationary cost incurred while in state s and the expected moving cost incurred in leaving state s. Since $\alpha \leq 1$, by Corollary 5.1 the algorithm cannot remain in s when $\Phi_s = 1$, and therefore we need only consider possible costs incurred by the algorithm during each period. In our analysis we think of the algorithm moving just before Φ_s reaches α, so we include the cost of moving in the cost of the period. Thus, we divide the time line of each state s into maximal periods. This is possible since Φ_s is piecewise linear.

We prove the competitiveness of the algorithm by establishing an upper-bound on the cost incurred by the algorithm during each of these periods and then summing over all periods. The upper bound for non-increasing periods is established in Lemma 5.3, and the bound for increasing periods is established in Lemma 5.4. Each lemma bounds the expected cost of the algorithm in terms of the increase in the useful work: if the useful work in a period increases by x, then the expected cost to the algorithm is at most x if it is a non-increasing period and $c \cdot x$ if it is an increasing period. (c is a constant in the range $1 < c < 2$). Lemma 5.2 states that if the useful work increases by x, we can attribute a cost of at least x/n to the adversary. These three bounds are put together in the proof of Theorem 5.1 to establish an upper bound of roughly $c \cdot n$ on the competitiveness of the algorithm.

Consider a single period of state s. We show that:

Lemma 5.3 *For any non-increasing period with respect to a state s, the ratio of the expected cost incurred by the* THRESHOLD WORK FUNCTION *algorithm while in state s or leaving state s to the increase in useful work is at most 1.*

and:

Lemma 5.4 *There exists a distribution $\mathcal{D}$ such that, for any increasing period with respect to a state s, the ratio of the expected cost incurred by the* THRESHOLD WORK FUNCTION *algorithm while in state s or leaving state s to the increase in useful work is at most c, for some constant c.*

Proof of Lemma 5.3 Let w be the work function at the beginning of the period, and w' be the work function at the end. Suppose that the work function of s increases from $x = w(s)$ to $x' = w'(s)$ during the period. If the given period is a non-increasing period with respect to s and the algorithm is in s, it incurs a cost of at most $x' - x$, since there is no possibility it moves from s. The useful work increases by $x' - x$, and the ratio of expected cost to increase in useful work is 1. ∎

Proof of Lemma 5.4 We begin by noting that during any increasing period with respect to a state s, the algorithm leaves s at most once during the period. To see this, note that the algorithm never visits any state s with $\Phi_s \geq \alpha$. If the algorithm is in s and Φ_s reaches α the algorithm leaves and does not return. The value of Φ_s achieves a value of α at most once during the period.

By definition, the value of $\eta(s)$ does not change during the period. Let t be the value of $\eta(s)$ during this period. Let w be the work function at the beginning of the period, and w' be the work function at the end. Let j be the least integer for which $\frac{j}{m} d_{s,t} \geq w(s) - w(t)$, and k be the greatest integer such that $\frac{k}{m} d_{s,t} \leq w'(s) - w'(t)$. Define:

$$\begin{array}{rcl} y_0 & = & w(s) - w(t), \\ y_1 & = & \frac{j}{m} d_{s,t}, \\ y_2 & = & \frac{j+1}{m} d_{s,t}, \\ \vdots & \vdots & \vdots \\ y_{k-j+1} & = & \frac{k}{m} d_{s,t}, \\ y_{k-j+2} & = & w'(s) - w'(t). \end{array}$$

Let t_i be the point in time, after the beginning of the period, at which $w(s) - w(t) = y_i$. Since the period is increasing, these times are well defined. The given period is broken into sub-periods, each of which is itself a period. The first sub-period starts at t_0 and ends at t_1, the second sub-period starts at t_1 and ends at t_2, etc. If the hypothesis is true for each of these sub-periods, then it is true for the period.

We analyze a single sub-period. Redefine w to be the work function at the beginning of the sub-period, and w' to be the work function at the end. Let $x = w(s)$ and $x' = w'(s)$. We define $y = w(s) - w(t)$ and $y' = w'(s) - w'(t)$. Note that $y' - y \leq x' - x$ since $w(t) \leq w'(t)$. The useful work increases by at least $x' - x$. Since we consider a single sub-period:

1. At the start of the sub-period, we have that $y > \frac{i}{m} d_{s,t}$.

2. At the end of the sub-period, we have $y' \leq \frac{i+1}{m} d_{s,t}$.

for some i. First consider the stationary cost incurred due to this sub-period. Since $\Pr\left[\alpha \leq \frac{i}{m}\right] = P_{1,i-1}$, the probability that the algorithm is in s is at most $1 - P_{1,i-1} = P_{i,m}$. Given that the algorithm is in s, it incurs a stationary cost of at most $x' - x$. The stationary cost incurred is therefore at most

$$(x' - x) P_{i,m}.$$

Now consider the moving cost. If the algorithm moves from s to t it incurs a cost $d_{s,t}$. By definition $\Pr\left[\alpha \in \left(\frac{i}{m}, \frac{i+1}{m}\right]\right] = p_i$. The distribution of α within $\left(\frac{i}{m}, \frac{i+1}{m}\right]$ is uniform, and so the probability that the algorithm moves to t is proportional to $y' - y$. When $y' = y + \frac{1}{m} d_{s,t}$ the probability that the algorithm moves is exactly p_i. Combining these facts, we find that the expected moving cost is

$$m(y' - y)p_i.$$

Combining the moving and stationary costs we get

$$\begin{aligned}
& (x' - x)P_{i,m} + m(y' - y)p_i \\
\leq \quad & (x' - x)P_{i,m} + m(x' - x)p_i \\
= \quad & m(x' - x)\left(\frac{P_{i,m}}{m} + p_i\right).
\end{aligned}$$

Let

$$z_i = \frac{P_{i,m}}{m} + p_i.$$

We choose the distribution $p_1, \ldots, p_m$ so that $z_1 = z_2 = \cdots = z_m$. Consider z_i and z_{i+1}

$$\begin{aligned}
z_i &= \frac{P_{i,m}}{m} + p_i, \\
z_{i+1} &= \frac{P_{i+1,m}}{m} + p_{i+1}.
\end{aligned}$$

Assuming that $z_i = z_{i+1}$, the difference of these two equations is

$$p_i + p_i \tfrac{1}{m} - p_{i+1} = 0.$$

And so

$$p_{i+1} = (1 + \tfrac{1}{m})p_i.$$

This recurrence has solution

$$p_{i+1} = (1 + \tfrac{1}{m})^i p_1.$$

To find p_1, we note that

$$\begin{aligned}
1 &= \sum_{i=1}^{m} p_i \\
&= p_1 \sum_{i=1}^{m} (1 + \tfrac{1}{m})^{i-1} \\
&= p_1 \sum_{i=0}^{m-1} (1 + \tfrac{1}{m})^i
\end{aligned}$$

$$
\begin{aligned}
&= p_1 \frac{(1+\frac{1}{m})^m - 1}{1+\frac{1}{m}-1} \\
&= p_1 \left(m(1+\tfrac{1}{m})^m - m \right).
\end{aligned}
$$

And so

$$p_1 = \frac{1}{m(1+\frac{1}{m})^m - m}.$$

Since z_i is independent of i, we abbreviate z_i by z. We have

$$z = z_1 = \frac{1}{m} + \frac{1}{m(1+\frac{1}{m})^m - m}.$$

The increase in useful work is $x' - x$ and so, the ratio of expected cost to increase in useful work is at most

$$\frac{mz(x'-x)}{x'-x} = 1 + \frac{1}{(1+\frac{1}{m})^m - 1}.$$

■

Theorem 5.1 *Let*

$$c = 1 + \frac{1}{(1+\frac{1}{m})^m - 1}.$$

Then algorithm Threshold Work Function *is* $(c \cdot n - c + 1)$*-competitive.*

Proof The total useful work during the entire sequence is U_ℓ. Let A_ℓ be the increase in useful work that occurs during non-increasing periods. The useful work incurred in increasing periods is $U_\ell - A_\ell$. By lemmas 5.3 and 5.4, the total expected cost to the algorithm is at most

$$c(U_\ell - A_\ell) + A_\ell.$$

By Lemma 5.2, the cost to the optimal algorithm at least

$$\frac{U_\ell}{n} - d_{\max}.$$

Before bounding the competitiveness of the algorithm, we establish one more bound on A_ℓ. Observe that if s is a state with minimum work function value, then $\Phi_s = 0$. Thus, a period in which the state with the minimum work function value is charged is a non-increasing period. Since A_ℓ is the increase in the total useful work during non-increasing periods, A_ℓ is at least as large as the final minimum value of the work function. Using Lemma 5.2, we know that

$$A_\ell \geq \frac{U_\ell}{n} - d_{\max}.$$

We are now ready to put these bounds together to bound the competitiveness of the THRESHOLD WORK FUNCTION. Denote the cost incurred by the THRESHOLD WORK FUNCTION algorithm on input σ by $\mathrm{cost}_{\mathrm{TWF}}(\sigma)$. For the algorithm to be $(c \cdot n - c + 1)$-competitive we must have for some constant k

$$\mathrm{cost}_{\mathrm{TWF}}(\sigma) \leq (c \cdot n - c + 1)\mathrm{cost}(\sigma) + k,$$

for all task sequences σ. Let $k = c \cdot n \cdot d_{\max} - 2c \cdot d_{\max}$. Then we have

$$\begin{aligned}
\mathrm{cost}_{\mathrm{TWF}}(\sigma) &\leq c(U_\ell - A_\ell) + A_\ell \\
&\leq c\left(U_\ell - \frac{U_\ell}{n} - d_{\max}\right) + \frac{U_\ell}{n} - d_{\max} \\
&= c \cdot U_\ell - c\frac{U_\ell}{n} - c \cdot d_{\max} + \frac{U_\ell}{n} - d_{\max} \\
&= \left(c - \tfrac{c}{n} + \tfrac{1}{n}\right) U_\ell - c \cdot d_{\max} - d_{\max} \\
&\leq (c \cdot n - c + 1)\mathrm{cost}(\sigma) + k
\end{aligned}$$

■

Is this ratio of $c \cdot n - c + 1$ lower than the known ratio of $2n - 1$ [BLS92]? The competitive ratio of the algorithm is

$$\left(1 + \frac{1}{(1+\frac{1}{m})^m - 1}\right) n - \frac{1}{(1+\frac{1}{m})^m - 1}.$$

Note that $\lim_{m\to\infty} \left(1 + \frac{1}{m}\right)^m = e$. In fact, we have

$$\left(1 + \tfrac{1}{m}\right)^m = e - \frac{e}{2m} + \frac{11e}{24m^2} - \frac{7e}{16m^3} + O\left(\frac{1}{m^4}\right).$$

The competitive ratio can therefore be made arbitrarily close to

$$\left(1 + \frac{1}{e-1}\right) n - \frac{1}{e-1} = \left(\frac{e}{e-1}\right) n - \frac{1}{e-1}, \tag{1}$$

by choosing sufficiently large m. For large m the competitive ratio is approximately $1.581977n - 0.581977$.

It is interesting to note that, with $m = 1$, the distribution is uniform and the algorithm achieves a competitive ratio of $2n - 1$.

6 Conclusions

There are several open questions that remain pertaining to the specific problems we address in this paper. With respect to THRESHOLD WORK FUNCTION algorithm, we were unable to obtain a closed form expression for the continuous distribution over the algorithm's state which results from letting m go to infinity. The lower bound shown in the appendix only demonstrates that our analysis is tight as m goes to infinity. Another possibility is to consider the algorithm which moves from state s to state t when

$$w(s) = \alpha \cdot w(t) + \beta \cdot d_{s,t}$$

for randomly chosen α and β. Analysis of this algorithm might prove fruitful. More generally, although the THRESHOLD WORK FUNCTION algorithm is the first algorithm to improve upon the optimal deterministic algorithm for all metric spaces, there are specific metric spaces on which it performs very poorly with respect to the best known randomized algorithm. For example, as shown in this and previous papers, the uniform metric space has an $O(\log n)$-competitive algorithm. An outstanding open problem in this area is whether there is an algorithm with logarithmic (or at least sub-linear) competitive ratio for all metric spaces. The linear upper bound shown here (although the best that is known for general metric spaces) is very far from the current best lower bound of $\sqrt{\log n / \log\log n}$ in [BKRS92].

Metrical task systems have prompted the development of a variety of principles which have been of use in approaching other online problems. Certainly, it is our hope that the techniques developed here will be useful elsewhere. However, metrical task systems have more significance than simply as a source of difficult problems to inspire the development of technical tools. The model was originally developed in order to give insight to and provide a systematic approach for online applications. In all of this work, we have been assuming that the sequence consists of tasks which are arbitrary non-negative vectors. The generality of the input sequence severely limits how well an online algorithm performs in the worst case. To accurately model many online problems, the input sequence must be limited so that the tasks are drawn from a limited set of allowable tasks. Thus, in the more general definition, a task system is described by the underlying metric space and the set of allowable tasks. Borodin *et al.* [BLS92] recognized the need to investigate how different restrictions on the set of allowable tasks affect the competitive ratio. This direction is investigated further in [BI95]. Since this generalized definition is able to model a large class of online problems much more accurately, some of the most important open questions relating to metrical task systems lie in the area of designing algorithms for metrical tasks systems under this more general definition.

References

[BDBK+94] S. Ben-David, A. Borodin, R. Karp, G. Tardos, et al. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, Jan 1994.

[BI95] W. Burley and S. Irani. On algorithm design for metrical task systems. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete*

Algorithms (SODA), pages 420 – 429, 1995. To appear in Algorithmica.

[BKRS92] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem and bounds for randomized server problems. In *Proc. 33th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 197–207, 1992.

[BLS92] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task systems. *Journal of the ACM*, 39(4):745–763, Oct 1992.

[Bur93] W. R. Burley. Traversing layered graphs using the work function algorithm. Technical Report CS93-319, University of California, San Diego, Sep 1993.

[CL91] M. Chrobak and L. L. Larmore. Server problems and on-line games. In *Proc. DIMACS Workshop on On-line Algorithms*, pages 11–64, Feb 1991.

[CN] M. Chrobak and J. Noga. Unpublished manuscript.

[IS95] S. Irani and S. S. Seiden. Randomized algorithms for metrical task systems. In *Proceedings of the 4th Workshop on Algorithms and Data Structures*, pages 159–170. Springer-Verlag, Aug 1995. Published as LNCS 955.

[KMMO94] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. *Algorithmica*, 11(6):542–571, Jun 1994.

[MMS90] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[ST85a] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, Feb 1985.

[ST85b] D.D. Sleator and R.E. Tarjan. Self adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, Jul 1985.

[Wes92] J. Westbrook. Randomized algorithms for multiprocessor page migration. In *Proc. DIMACS Workshop on On-line Algorithms*, 1992.

Appendix

6.1 Lower Bound for the Threshold Work Function Algorithm

Consider the following adversarial strategy for the uniform distance metric: The adversary breaks the task sequence into *phases*. At the beginning of phase i, the work function of every state reaches i. At the end of the phase i, the work function of every state reaches $i+1$ (and the phase $i+1$ begins). We further break the phase into *sub-phases*. A sub-phase ends whenever some state reaches $i+1$. There is one sub-phase during phase 0, since $w_0(s) = 1$ for all $s \neq s_1$. There are n sub-phases in each subsequent phase; the adversary charges each state from i to $i+1$, one by one. Let $U_{i,j}$ be the set of states u such that $w(u) < i+1$ at the beginning of sub-phase j of the phase i. We consider the cost to the algorithm of phase i, where $i > 0$.

Consider sub-phase j where $j \neq n$ (i.e. it is not the last sub-phase). Suppose that at the beginning of this sub-phase, the adversary knows (with absolute certainty) which state s the algorithm is in. We have that $s \in U_{i,j}$. The adversary charges s from i to $i+1$. Note that during the sub-phase for all $u, v \in U_{i,j}$ with $u \neq s$ and $v \neq s$ we have $\phi_s(u) = \phi_s(v)$. The algorithm moves to some other state $s' \neq s$ in $U_{i,j}$. The adversary knows to which state the algorithm moves, since the algorithm breaks ties deterministically. (If the algorithm breaks ties randomly, the adversary forces the algorithm to move to a chosen state by charging all other states infinitesimally.) The algorithm incurs a moving cost of 1 for the sub-phase. The stationary cost is $\mathrm{E}[\alpha]$. By the choice of α we have

$$
\begin{aligned}
\mathrm{E}[\alpha] &= \sum_{i=1}^{m} p_i \frac{i-\frac{1}{2}}{m} \\
&= \sum_{i=1}^{m} p_1 \frac{2i-1}{2m}(1+\tfrac{1}{m})^{i-1} \\
&= \frac{p_1}{2m} \sum_{i=1}^{m} (2i-1)(1+\tfrac{1}{m})^{i-1} \\
&= \frac{p_1}{2m} \sum_{i=0}^{m-1} (2i+1)(1+\tfrac{1}{m})^{i} \\
&= \frac{p_1}{2m} \left(2\sum_{i=0}^{m-1} i(1+\tfrac{1}{m})^{i} + \sum_{i=0}^{m-1} (1+\tfrac{1}{m})^{i} \right) \\
&= \frac{p_1}{2m} \Big(2m^2 \left((m-1)(1+\tfrac{1}{m})^{m+1} - m(1+\tfrac{1}{m})^{m} + 1 + \tfrac{1}{m} \right) + \\
&\qquad m \left((1+\tfrac{1}{m})^{m} - 1 \right) \Big)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{p_1}{2}\Big(2m(m-1)(1+\tfrac{1}{m})^{m+1} - 2m^2(1+\tfrac{1}{m})^m + 2m + 2 + \\
&\qquad (1+\tfrac{1}{m})^m - 1\Big) \\
&= \frac{p_1}{2}\left((2m^2-2m)(1+\tfrac{1}{m})(1+\tfrac{1}{m})^m - (2m^2-1)(1+\tfrac{1}{m})^m + 2m + 1\right) \\
&= \frac{p_1}{2}\left((2m^2-2)(1+\tfrac{1}{m})^m - (2m^2-1)(1+\tfrac{1}{m})^m + 2m + 1\right) \\
&= \frac{p_1}{2}\left(2m - (1+\tfrac{1}{m})^m + 1\right) \\
&= \frac{2m - (1+\frac{1}{m})^m + 1}{2m(1+\frac{1}{m})^m - m} \\
&= \frac{1}{(1+\frac{1}{m})^m - 1} + \frac{1-(1+\frac{1}{m})^m}{2m\left((1+\frac{1}{m})^m - 1\right)} \\
&= \frac{1}{(1+\frac{1}{m})^m - 1} - \frac{1}{2m}.
\end{aligned}
$$

Hence, counting also the moving cost, the total cost to the algorithm for the sub-phase is $1 + \mathrm{E}[\alpha]$.

During the last sub-phase, the algorithm incurs a stationary cost of 1 and incurs no moving cost.

For the phase, the algorithm incurs a total cost of $(n-1)(1+\mathrm{E}[\alpha]) + 1 = (1+\mathrm{E}[\alpha])n - \mathrm{E}[\alpha]$. The adversary incurs a cost of 1. So the competitive ratio of the algorithm is at least $(1+\mathrm{E}[\alpha])n - \mathrm{E}[\alpha]$. For large m, this lower bound on the competitive ratio approaches $\frac{e}{e-1}n - \frac{1}{e-1}$.

6.2 Analysis of the Alpha Work Function Algorithm

The proof is very similar in structure to the proof in [BLS92]. Let $\mathrm{cost}^k_{\mathrm{AWF}}$ be the cost incurred by the Alpha Work Function algorithm in processing the first k tasks. Let cost^k be the optimal offline cost of processing the first k tasks. To begin, we need three lemmata. The first two are shown by Borodin, Linial and Saks.

Lemma 6.1 $\mathrm{cost}^k \geq \min_{s \in S} w_k(s)$, *for all* $s \in S$. ∎

Lemma 6.2 $w_k(s) - w_k(t) \leq d_{s,t}$, *for all* $s, t \in S$. ∎

Let $\beta = 1 + \frac{1}{\alpha}$ and

$$W_k = w_k(\pi(k)) + \beta \sum_{s \neq \pi(k)} w_k(s).$$

We use the following abbreviations:

$$w = w_{k-1}, \quad w' = w_k, \quad W = W_{k-1}, \quad W' = W_k, \quad x = \pi(k-1), \quad x' = \pi(k).$$

Lemma 6.3 *For all k:*

$$W_k \geq \mathrm{cost}^k_{\mathrm{AWF}}.$$

Proof Note that if the algorithm services the task k in state x' then $w'(x') = w(x') + T^k(x')$. Consider the cost to the algorithm for task k. There are two possibilities:

1. Suppose that the algorithm does not move to service task k. Then $x' = x$. The cost to the algorithm for this task is just $T^k(x')$, since there is no moving cost. We have that

$$w'(x') - w(x') = T^k(x').$$

   We calculate the change in the value of W

$$\begin{aligned}
\Delta W' &= W' - W \\
&= w'(x') + \beta \sum_{s \neq x'} w'(s) - \Big(w(x) + \beta \sum_{s \neq x} w(s) \Big) \\
&= \beta \Big(\sum_{s \neq x'} w'(s) - \sum_{s \neq x} w(s) \Big) + w'(x') - w(x) \\
&= \beta \sum_{s \neq x'} \big(w'(s) - w(s) \big) + w'(x') - w(x') \\
&\geq w'(x') - w(x') \\
&= T^k(x'),
\end{aligned}$$

   which is just the cost to the algorithm for task k.

2. On the other hand, if the algorithm moves then $x' \neq x$ and its cost is

$$T^k(x') + d_{x,x'}.$$

   Once again, we calculate the change in the value of W

$$\begin{aligned}
\Delta W' &= W' - W \\
&\geq \beta w'(x) + w'(x') - \big(\beta w(x') + w(x) \big).
\end{aligned}$$

   Since each work function increases with time, and we are merely lower bounding $\Delta W'$ by the increase of two of these work functions. We have

$$\begin{aligned}
\Delta W' &\geq \beta w'(x) + w'(x') - \big(\beta w(x') + w(x) \big) \\
&= \tfrac{1}{\alpha} w'(x) + w'(x') - \beta w(x') + w'(x) - w(x)
\end{aligned}$$

$$
\begin{aligned}
&\geq \tfrac{1}{\alpha}w'(x) + w'(x') - \beta w(x') \\
&= \tfrac{1}{\alpha}\big(w'(x) - w(x')\big) + \big(w'(x') - w(x')\big) \\
&\geq \tfrac{1}{\alpha}\big(w'(x) - w'(x')\big) + \big(w'(x') - w(x')\big) \\
&\geq d_{x',x} + T^k(x'),
\end{aligned}
$$

which, again, is just the cost to the algorithm for task k. ∎

We prove the main result.

Theorem 6.1 *The Alpha Work Function Algorithm is $\big((1+\frac{1}{\alpha})n-\frac{1}{\alpha}\big)$-competitive for $\alpha \leq 1$, i.e.*

$$\text{cost}^k \leq \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big)\text{cost}^k_{\text{AWF}} + b,$$

for all request sequences of length k, for some constant b independent of k.

Proof By Lemma 6.1 we have that $\text{cost}^k \geq \min_{s\in S} w'(s)$. Let s^* be the state which minimizes $w'(s)$, i.e. $w'(s^*) = \min_{s\in S} w'(s)$. Let $d_{\max}$ be the maximum distance between two states. We prove the theorem by bounding W'. By Lemma 6.2 we get

$$
\begin{aligned}
W' &\leq \beta \sum_{s\neq x'} \big(w'(s^*) + d_{s,s^*}\big) + w'(s^*) + d_{x',s^*} \\
&= \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big) \min_{s\in S} w'(s) + \beta \sum_{s\neq x'} d_{s,s^*} + d_{x',s^*} \\
&\leq \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big) \min_{s\in S} w'(s) + \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big)d_{\max} \\
&\leq \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big)\text{cost}^k + \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big)d_{\max}.
\end{aligned}
$$

By Lemma 6.3 we have that $\text{cost}^k_{\text{AWF}} \leq W'$ and so

$$\text{cost}^k_{\text{AWF}} \leq \big((1+\tfrac{1}{\alpha})n - \tfrac{1}{\alpha}\big)\text{cost}^k + b,$$

for some constant b. ∎