An Optimal On-Line Algorithm for Metrical Task System

ALLAN BORODIN

University of Toronto, Toronto, Ontario, Canada

NATHAN LINIAL

Hebrew University, Jerusalem, Israel, and IBM Almaden Research Center, San Jose, California

AND

MICHAEL E. SAKS

University of California at San Diego, San Diego, California

Abstract. In practice, almost all dynamic systems require decisions to be made on-line, without full knowledge of their future impact on the system. A general model for the processing of sequences of tasks is introduced, and a general on-line decision algorithm is developed. It is shown that, for an important class of special cases, this algorithm is optimal among all on-line algorithms.

Specifically, a task system (S, d) for processing sequences of tasks consists of a set S of states and a cost matrix d where $d(i, j)$ is the cost of changing from state i to state j (we assume that d satisfies the triangle inequality and all diagonal entries are 0). The cost of processing a given task depends on the state of the system. A schedule for a sequence $T^1, T^2, \ldots, T^k$ of tasks is a sequence $s_1, s_2, \ldots, s_k$ of states where s_i is the state in which T^i is processed; the cost of a schedule is the sum of all task processing costs and state transition costs incurred.

An on-line scheduling algorithm is one that chooses s_i only knowing $T^1 T^2 \cdots T^i$. Such an algorithm is w-competitive if, on any input task sequence, its cost is within an additive constant of w times the optimal offline schedule cost. The competitive ratio $w(S, d)$ is the infimum w for which there is a w-competitive on-line scheduling algorithm for (S, d). It is shown that $w(S, d) = 2|S| - 1$ *for every task system* in which d is symmetric, and $w(S, d) = O(|S|^2)$ for every task system. Finally, randomized on-line scheduling algorithms are introduced. It is shown that for the uniform task system (in which $d(i, j) = 1$ for all i, j), the expected competitive ratio $\bar{w}(S, d) = O(\log |S|)$.

A. Borodin was supported in part while he was a Lady Davis visiting professor at the Hebrew University in Jerusalem. M. E. Saks was supported in part by National Science Foundation grants DMS 87-03541, CCR 89-11388 and Air Force Office of Scientific Research grant AFOSR-0271.

Authors' addresses: A. Borodin, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4; N. Linial, Institute of Computer Science and Mathematics, Hebrew University, Jerusalem, Israel; M. E. Saks, Department of Computer Science, University of California at San Diego, San Diego, CA.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*; G.3 [**Probability and Statistics**]: *probabilistic algorithms (including Monte Carlo)*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Competitive analysis, on-line algorithms

1. *Introduction*

Computers are typically faced with streams of tasks that they have to perform. It is usually the case that a task can be performed in more than one way. For example, the system may have several possible paging schemes available or the same piece of data may reside in several files each of which can be accessed. The decision as to how to perform a particular task affects the efficiency of the system in two ways: (1) the cost of the present task depends on how it is performed and (2) the state of the system upon completion of the task may affect the cost of subsequent tasks.

There are many other situations in which present actions influence our well being in the future. In economics, one often encounters situations in which the choice between present alternatives influences the future costs/benefits. This is, for example, the case in choosing an investment strategy, for example, in selecting among investment plans with different interest rates and time spans.

The difficulty is, of course, that we have to make our decision based only on the past and the current task we have to perform. It is not even clear how to measure the quality of a proposed decision strategy. The approach usually taken by mathematical economists is to devise some probabilistic model of the future and act on this basis. This is the starting point of the theory of Markov decision processes [13].

The approach we take here, which was first explicitly studied in the seminal paper of Sleator and Tarjan [21], is to compare the performance of a strategy that operates with no knowledge of the future with the performance of the optimal "clairvoyant" strategy that has complete knowledge of the future. This requires no probabilistic assumptions about the future and is therefore a "worst-case" measure of quality. A number of recent articles that preceded this paper analyze various computational problems from this point of view [2, 7, 15, 19, 22], and many more papers have appeared subsequently.

More precisely, consider a system for processing tasks that can be configured into any of a possible set S of states (we take S to be the set $\{1, 2, \ldots, n\}$). For any two states $i \neq j$, there is a positive transition cost $d(i, j)$ associated with changing from i to j. We always assume that the distance matrix d satisfies the triangle inequality, $d(i, j) + d(j, k) \geq d(i, k)$ and that the diagonal entries are 0. The pair (S, d) is called a *task system*. For any task T, the cost of processing T is a function of the state of the system so we can view T as a vector $T = (T(1), T(2), \ldots, T(n))$ where $T(j)$ is the (possibly infinite) cost of processing the task while in state j. An instance $\mathbf{T}$ of the scheduling problem consists of a pair $\{T^1T^2 \cdots T^m, s_0\}$ where $T^1T^2 \cdots T^m$ is a sequence of tasks and s_0 is a specified initial state. $\mathbf{T}$ is called a *task sequence* and usually we write simply $\mathbf{T} = T^1T^2 \cdots T^m$, where it is understood that there is a specified initial state s_0. (For most of what we do, the particular value of s_0 is unimportant.) A *schedule* for processing a sequence $\mathbf{T} = T^1T^2 \cdots T^m$ of tasks is a function $\boldsymbol{\sigma}$: $\{0, \ldots, m\} \to S$ where $\boldsymbol{\sigma}(i)$ is the state in which T^i is processed (and $\boldsymbol{\sigma}(0) = s_0$).

The cost of schedule $\boldsymbol{\sigma}$ on $\mathbf{T}$ is the sum of all state transition costs and the task processing costs:

$$c(\mathbf{T};\boldsymbol{\sigma}) = \sum_{i=1}^{m} d(\boldsymbol{\sigma}(i-1),\boldsymbol{\sigma}(i)) + \sum_{i=1}^{m} T^i(\boldsymbol{\sigma}(i)).$$

It will be convenient to view the ith task as arriving at time i and being processed during the time interval $[i, i+1)$. (Thus, the first task arrives at time 1.) A schedule $\boldsymbol{\sigma}$ is also specified by sequences of state transitions and transition times. Specifically $1 \le t_1 < t_2 < \cdots < t_k$ is the sequence of times at which a state change occurs and $s = s_1, s_2, \ldots, s_k$ is the order in which states are visited (the states are not necessarily distinct but $s_{j-1} \neq s_j$ for each j). We also define $t_0 = 0$.

A *scheduling algorithm* for (S, d) is a map A that associates to each task sequence $\mathbf{T}$ a schedule $\boldsymbol{\sigma} = A(\mathbf{T})$. The cost of algorithm A on sequence $\mathbf{T}$, denoted $c_A(\mathbf{T})$ is defined to be $c(\mathbf{T}; A(\mathbf{T}))$. It is easy to construct a dynamic programming algorithm that gives an optimal (minimum cost) schedule for any task sequence; we denote this optimal cost by $c_0(\mathbf{T})$. This algorithm is an off-line computation since the choice of each state may depend on future tasks.

In an *on-line scheduling algorithm*, the ith state $\boldsymbol{\sigma}(i)$ of the resulting schedule $\boldsymbol{\sigma}$ is a function only of the first i tasks $T^1T^2 \cdots T^i$ and s_0; that is, the tasks are received one at a time in order and the algorithm must output $\boldsymbol{\sigma}(i)$ upon receiving T^i. To measure the efficiency of an on-line algorithm A as compared to the optimal (off-line) algorithm, we say that A is *w-competitive* (after [15]) for a positive real number w if there is a constant K_w such that $c_A(\mathbf{T}) - wc_0(\mathbf{T}) \le K_w$ for any finite task sequence $\mathbf{T}$. Let $W_A = \{w$: *A is w-competitive*$\}$ and define the *competitive ratio* $w(A)$ to be the infimum of W_A. The *competitive ratio* of the task system, $w(S, d)$, is the infimum of $w(A)$ over all on-line algorithms A.[1]

If, in addition to satisfying the triangle inequality, the transition cost matrix is symmetric, that is, $d(i, j) = d(j, i)$, we say that the task system (S, d) is *metrical*. Our main result is that the competitive ratio of every metrical task system on n states is independent of d:

THEOREM 1.1. *For any metrical task system (S, d) with n states*

$$w(S, d) = 2n - 1.$$

In many situations, the matrix d is not symmetric since the cost of switching between two states may be more expensive one way than the other. In fact, the algorithm we give can be used to give an upper bound in the general case. For a matrix d that is not necessarily symmetric, define the cycle offset ratio $\psi(d)$ to be the maximum over all sequences $s_0, s_1, s_2, \ldots, s_k$ where $s_0 = s_k$ of the ratio $(\sum_{i=1}^{k} d(s_{i-1}, s_i))/(\sum_{i=1}^{k} d(s_i, s_{i-1}))$. (If d is symmetric, then $\psi(d) = 1$.) We show

THEOREM 1.2. *For any task system (S, d) with n states, there is an on-line algorithm A_d^* with competitive ratio at most $(2n-1)\psi(d)$.*

Since d satisfies the triangle inequality, it is easy to show that $\psi(d) \le n - 1$, and so $w(S, d) \le (n-1)(2n-1)$ for any task sequence. We do not believe

[1] In our original paper, we used the term, *waste factor*, rather than the now accepted terminology of *competitive ratio*.

that this is the best possible; our guess is that an $O(n)$ competitive ratio is always achievable.

The work in this paper was motivated by the goal of developing a general theory of on-line algorithms for sequential decision problems. Let us explain how it relates to some of the work done in particular cases, for instance, sequential search of a list. In this case, on-line algorithms have been developed that give a competitive ratio of at most 2 [21]. How does this square with our result, that there is an $\Omega(|S|)$ lower bound on the competitiveness for any metrical task system? The difference is that in the sequential search problem and in other particular problems, the possible tasks are restricted to a very special set, while in our model we allow arbitrary tasks. The next step in this research is to determine how restrictions on the set of tasks can effect the competitive ratio. In recent work, Manasse et al. [16] have studied such restrictions by extending the model so that a task system consists of (S, d) together with a set T of possible tasks. In particular, they obtain partial results on a very interesting problem called the k-server problem. In our conclusion, we elaborate further on the relation between task systems and the k-server problem.

In Section 2, the lower bound of $2n - 1$ for metrical task systems is proved. First, the situation is formulated as a game between an on-line scheduler and a taskmaster and then a particular strategy for the taskmaster is shown to force a competitive ratio of $2n - 1$. The upper bounds of Theorems 1.1 and 1.2 are proved by explicit description of on-line algorithms. These algorithms are most easily described as operating in continuous time (allowing nonintegral transition times). In Section 3, such algorithms are defined precisely and shown to be easily converted to discrete time algorithms. In Section 4, a simple class of algorithms called nearly oblivious algorithms is introduced, whose behavior depends only very weakly on the input task system. In Section 5, a simple algorithm that achieves a competitive ratio of at most $8(n - 1)$ when d is symmetric is presented. In Section 6, the algorithm promised by Theorem 1.2 is given. In Section 7, randomized on-line algorithms are introduced and a special case is analyzed. Section 8 contains some concluding remarks and comparisons to other work.

2. *Cruel Taskmasters and a Lower Bound*

We can view the operation of an on-line algorithm as a game between the *taskmaster* and the *scheduler*; in the ith round the taskmaster provides the task T^i and then the scheduler chooses $\sigma(i)$. To prove a lower bound we will define a simple set of strategies for the taskmaster and show that they can be used to force a competitive ratio arbitrarily close to $2n - 1$.

For a real number $\epsilon > 0$, we define the *cruel taskmaster strategy* $M(\epsilon)$, as follows: At step i, define T^i so that

$$T^i(\sigma(i-1)) = \epsilon,$$
$$T^i(s) = 0 \qquad \text{if} \quad s \neq \sigma(i-1).$$

Thus, the cost of the ith task is nonzero if and only if the scheduler remains in state $\sigma(i - 1)$. Such tasks are called ϵ-*elementary.*

To prove a lower bound of $2n - 1$ on the competitive ratio of an arbitrary algorithm A, it will be convenient to define the competitive ratio of A with respect to an infinite task sequence $\mathbf{T}$ as follows:

$$w_{\mathbf{T}}(A) = \limsup_{m\to\infty} \frac{c_A(T^1T^2 \cdots T^m)}{c_0(T^1T^2 \cdots T^m)}.$$

From the definition of $w(A)$, it is easy to prove:

LEMMA 2.1. *If* $\mathbf{T}$ *is an infinite task sequence such that* $c_0(T^1T^2 \cdots T^m)$ *tends to infinity with* m, *then* $w(A) \geq w_{\mathbf{T}}(A)$.

PROOF. Let

$$r_m = c_A(T^1T^2 \cdots T^m)$$

and

$$g_m = c_0(T^1T^2 \cdots T^m).$$

It is enough to show that for all $w \in W_A$, $w \geq w_{\mathbf{T}}(A)$. So suppose $w \in W_A$; then there is a constant K_w such that $r_m - wg_m \leq K_w$ for all m, which implies:

$$\frac{r_m}{g_m} \leq w + \frac{K_w}{g_m}.$$

Taking lim sup of both sides yields $w_{\mathbf{T}}(A) \leq w$ since g_m tends to infinity, by hypothesis. □

The lower bound on $w(S, d)$ now follows from

THEOREM 2.2. *Let* A *be any on-line scheduling algorithm for the metrical task system* (S, d). *Let* $\mathbf{T}(\epsilon)$ *be the infinite task sequence produced by* $M(\epsilon)$ *in response to* A. *Then*

$$w_{\mathbf{T}(\epsilon)}(A) \geq \frac{(2n-1)}{(1 + \epsilon/min_{i\neq j}\, d(i,j))}.$$

Taking ϵ to be arbitrarily small, we obtain, by Lemma 2.1, that $w(A) \geq 2n - 1$ for any on-line algorithm A so $w(S, d) \geq 2n - 1$, as desired.

Remark. Intuitively, the reason that the lower bound on the competitive ratio improves as ϵ decreases is that smaller tasks reduce the amount of information available to the on-line algorithm at the times it must make decisions. Compare a sequence of k repeats of the task T with a single task kT (i.e., the task whose costs are obtained from those of T on multiplying by k). The cost for the off-line algorithm is easily seen to be the same in both cases. The on-line algorithm fares better with kT: Its situation is the same as with T repeated k times except that it is "given a guarantee" that the following k tasks are all equal to T.

PROOF OF THEOREM 2.2. Let $T^1T^2 \cdots$ be the (infinite) sequence of tasks produced by $M(\epsilon)$ and let σ be the schedule produced by A. Each ϵ-elementary task T^i has nonzero cost only in state $\sigma(i-1)$. Let $s_1, s_2, s_3, \ldots$ and $t_1 < t_2 < t_3 < \cdots$ be the state transitions and transition times prescribed by algorithm A and $t_0 = 0$. Recall from the introduction that we assumed that $\sigma(0)$ is occupied during the interval $[0, 1)$, and the first task arrives at time $t_0' = 1$. All the times t_i and t_i' are positive integers. Note that the times

$t_1' < t_2' < \cdots$ with $t_k' = t_k + 1$ define the times at which the taskmaster changes tasks.

To evaluate the competitive ratio, we analyze the quantities:

$$a_k = \text{cost incurred by } A \text{ during } [1, t_k'),$$
$$b_k = \text{cost incurred by the optimal off-line algorithm during } [1, t_k'].$$

By the definitions, we do not include the cost of the on-line transition at time t_k' in a_k but we do include the cost of an off-line transition at time t_k' in b_k. This discrepancy makes the analysis more convenient and, since we are ultimately interested in the ratio a_k/b_k as k tends to infinity, it does not affect its correctness. Note first that $a_k = D_k + P_k$ where

$$D_k = \sum_{i=1}^{k} d(s_{i-1}, s_i)$$

is the cost of state transitions for A and

$$P_k = \sum_{i=1}^{t_k} T'(\sigma(i)) = \sum_{i=1}^{t_k} \begin{Bmatrix} \epsilon & \text{if} \quad \sigma(i) = \sigma(i-1) \\ 0 & \text{if} \quad \sigma(i) \neq \sigma(i-1) \end{Bmatrix},$$
$$= \epsilon(t_k - k),$$

is the task processing cost for A.

We obtain an upper bound for b_k, as follows: Let $b_k(s)$ denote the cost of the optimal off-line schedule during the interval $[1, t_k']$ subject to the condition that the system is in state s at time t_k'. Then

$$b_k = \min_{s \in S} b_k(s),$$

so we can upper bound b_k by any convex combination of the $b_k(s)$. Thus, our goal is to find some linear combination B_k of the $b_k(s)$ for which we can write a bound in terms of D_k and P_k. To this end, we note the following facts:

$$b_k(s) = b_{k-1}(s) \qquad s \neq s_{k-1},$$
$$b_k(s_{k-1}) \leq \min\{b_{k-1}(s_{k-1}) + \epsilon(t_k - t_{k-1}), b_{k-1}(s_k) + d(s_k, s_{k-1})\}$$
$$\leq \frac{1}{2}\{b_{k-1}(s_{k-1}) + b_k(s_k) + \epsilon(t_k - t_{k-1}) + d(s_k, s_{k-1})\}.$$

The first claim is obvious, since staying in state s adds no cost between time t_{k-1}' and t_k'. The second follows from the fact that two possible options of the off-line algorithm are: (i) stay in state s_{k-1} during the interval $[t_{k-1}', t_k']$ and (ii) stay in state s_k during the interval $[t_{k-1}', t_k')$ and move to s_{k-1} at time t_k'.

Simple manipulation now shows that if we define

$$B_k = 2 \sum_{s \neq s_k} b_k(s) + b_k(s_k),$$

then

$$B_k \leq B_{k-1} + d(s_{k-1}, s_k) + \epsilon(t_k - t_{k-1}),$$

and thus

$$B_k \le B_1 + D_k + \epsilon t_k = B_1 + a_k + \epsilon k.$$

Now, since $a_k \ge D_k \ge k \min_{i \ne j} d(i,j)$ we have $\epsilon k \le \epsilon a_k / \min_{i \ne j} d(i,j)$. Also $B_k \ge (2n-1) b_k$ and thus the previous inequality implies:

$$b_k(2n-1) \le B_1 + a_k \left(1 + \frac{\epsilon}{\min_{i \ne j} d(i,j)} \right).$$

Simple manipulation yields

$$\frac{a_k}{b_k} \ge (2n-1) \left(\frac{1}{1 + \epsilon / \min d(i,j)} \right) - \frac{C}{b_k}$$

for some constant C. Taking the lim sup as k tends to ∞,

$$w_{\mathrm{T}(\epsilon)}(A) \ge (2n-1) \left(\frac{1}{1 + \epsilon / \min_{i \ne j} d(i,j)} \right).$$

3. *Continuous Time Schedules*

Because of the discrete nature of our problem, all state transitions occur at integer times. It will be useful to consider schedules in which transition times are allowed to be arbitrary nonnegative real numbers. Such continuous-time schedules are not really possible in our model, but we see in Lemma 3.1 that any such schedule can be converted easily to a discrete time schedule that is at least as good. The advantage of introducing continuous time is that the algorithms we define in the next sections are easier to describe.

Precisely, a *continuous-time schedule* for tasks $T^1 T^2 \cdots T^m$ is a function $\sigma : [1, m+1) \to S$ such that for each state $s \in S$, $\sigma^{-1}(s)$ is a finite disjoint union of half open intervals $[a, b)$. The endpoints of these intervals represent times for transitions between states. As before, σ can be described by the sequences $s_1, s_2, \ldots, s_k$ and $t_1 < t_2 < \cdots < t_k$ where state s_i is entered at time t_i; the only difference is that the t_i need not be integers. The cost of processing task T^i under this schedule is given by

$$\int_i^{i+1} dt\, T^i(\sigma(t)),$$

that is, the cost is the convex combination $\sum \lambda_s T^i(s)$ where λ_s is the proportion of the time interval spent in state s. The total cost of the schedule is

$$\sum_{j=1}^{k} d(s_{j-1}, s_j) + \sum_{i=1}^{m} \int_i^{i+1} dt\, T^i(\sigma(t)).$$

An on-line *continuous-time scheduling algorithm* (CTSA) is one that schedules the time interval $[i, i+1)$ upon receipt of task T^i. The algorithms we describe in the next three sections are CTSAs. We now show that any such algorithm can be converted into an online *discrete-time scheduling algorithm* (DTSA) that is at least as good on any task sequence.

LEMMA 3.1. *For any online CTSA A', there is an online DTSA A that performs at least as well as A on any task sequence* $\mathbf{T}$.

PROOF. We define A as follows: On task sequence $\mathbf{T}$, the schedule $\boldsymbol{\sigma}$ produced by A is given by

$$\boldsymbol{\sigma}(j) = s\colon T^j(s) \text{ is minimum among } s \in S_j$$

where S_j is the set of states visited by algorithm A' during the time interval $[j, j+1)$. Since A' is on-line, $\boldsymbol{\sigma}(j)$ is computable from s_0 and $T^1T^2 \cdots T^j$. Furthermore, we claim that the cost incurred by algorithm A is at most the cost incurred by A'. This follows from

(1) By the definition of $\boldsymbol{\sigma}(j)$, the sequence of states visited by A up to time j is a subsequence of the states visited by A'. Hence, by the triangle inequality, the total transition cost for A is at most that for A'.
(2) The cost of processing task j in A' is a convex combination of $T^j(s)$ for $s \in S_j$ and hence is at least $T^j(\boldsymbol{\sigma}(j))$. □

4. *Nearly Oblivious Algorithms and Traversal Algorithms*

For motivation, consider a very simple metrical task system consisting of two states s_0 and s_1. A "safe" on-line continuous algorithm A is as follows: A remains in starting state s_0 until the task processing cost incurred by A equals $d(s_0, s_1)$. A then moves to state s_1 where it stays until the task processing cost incurred by A in s_1, equals $d(s_1, s_0) = d(s_0, s_1)$. At this time, say time c, A returns to state s_0. This strategy of cycling between s_0 and s_1 now repeats. We note during the time interval $[0, c]$, that A incurs state transition costs of $2d(s_0, s_1)$ and task-processing costs of $2d(s_0, s_1)$; that is, a total cost of $4d(s_0, s_1)$. Consider the cost of any off-line algorithm B during the same time interval $[0, c]$ (and against the same sequence of tasks). If B makes a state transition, then B incurs a cost of at least $d(s_0, s_1)$. Otherwise, B remains in one of the states $s_i (i = 0$ *or* $i = 1)$ during the entire time interval, and this includes the time interval that A was in state s_i incurring a task-processing cost of at least $d(s_0, s_1)$. It follows then that whether or not B moves, A's cost is at most 4 times that of B.

This simple on-line algorithm has certain nice properties that can be abstracted as follows: An on-line CTSA is called *nearly oblivious* (so called because its behavior depends only weakly on the input task sequence) if it is specified by two infinite sequences $s_0 s_1 s_2 \cdots$ of states (s_0 is given), and $c_0 c_1 c_2 c_3 \cdots$ of positive reals, and produces a schedule as follows:

(1) The states are visited in the sequence $s_1, s_2, s_3, \cdots$ independent of the input task sequence.
(2) The transition from state s_j to s_{j+1} occurs at the instant in time t_{j+1} when the total processing cost incurred since entering s_j reaches c_j. Hence, t_{j+1} is defined so that

$$c_j = \int_{t_j}^{t_{j+1}} T^{[t]}(s_j)\, dt.$$

Such an algorithm clearly operates on-line. We say it is *periodic* with period k if, for $j \geq k$, $s_j = s_{j-k}$ and $c_j = c_{j-k}$. A periodic algorithm is a *traversal* *algorithm* if for each j, $c_j = d(s_j, s_{j+1})$, that is, the processing cost in state j is equal to the cost incurred in the transition to state s_{j+1}. A traversal algorithm (with period k) is completely specified by the sequence $\boldsymbol{\tau} = s_0 s_1 s_2 \cdots s_k (= s_0)$, which is called a *traversal* of S.

The traversal algorithm specified by the traversal $\boldsymbol{\tau}$ is called $A(\boldsymbol{\tau})$. A *cycle* of $A(\boldsymbol{\tau})$ on a given task sequence $\mathbf{T}$ is the time during which one traversal is completed. Clearly,

PROPOSITION 4.1. *If* $\boldsymbol{\tau} = s_0 s_1 s_2 \cdots s_k$, *then* $A(\boldsymbol{\tau})$ *incurs a cost of exactly* $2\sum_{i=1}^{k} d(s_{i-1}, s_i)$ *during each cycle.*

5. *A Good Traversal Algorithm*

We now generalize the two-state traversal algorithm discussed in the last section so as to achieve an $8(n-1)$ competitive ratio for any n state metrical task system. Intuitively, we partition the states into two components and we remain in a component (using the strategy recursively) until the task-processing cost incurred in that component equals the cost to make the transition to the other component. Formally, we proceed as follows: For d symmetric, let us view the transition costs $d(i, j)$ as edge weights on the (undirected) complete graph on n vertices. Let $MST = (S, E)$ be a minimum weight spanning tree for this graph. For $(u, v) \in E$, we define the *modified weight* $d'(u, v)$ to be 2^p where $2^{p-1} < d(u, v) \le 2^p$. We now inductively define a traversal $\boldsymbol{\tau}^*$ of S:

(a) If $|S| = 1$ and $S = \{u\}$, we have an empty traversal.
(b) For $|S| \ge 2$, let (u, v) be an edge with maximum weight in MST and let $d'(u, v) = 2^M$. The removal of (u, v) partitions the MST into two smaller trees $MST_1 = (S_1, E_1)$ and $MST_2 = (S_2, E_2)$ with $u \in S_1$ and $v \in S_2$. For $i = 1, 2$, let $\boldsymbol{\tau}_i$ be the inductively constructed traversals for S_i, and let 2^{M_i} be the maximum of the modified edge weights in E_i. Starting at u, $\boldsymbol{\tau}$ consists of 2^{M-M_1} cycles of $\boldsymbol{\tau}_1$, followed by the edge (u, v), followed by 2^{M-M_2} cycles of $\boldsymbol{\tau}_2$, followed by the edge (v, u). (Here we cyclically permute $\boldsymbol{\tau}_1$ (resp., $\boldsymbol{\tau}_2$) if necessary so that it begins and ends at u (resp., v).)

A simple induction yields:

LEMMA 5.1. *Let* 2^M *be the largest modified weight in MST. Then, in* $\boldsymbol{\tau}^*$, *an edge of modified weight* 2^m *is traversed* 2^{M-m} *times in each direction.*

From this, and Proposition 4.1, we have

COROLLARY 5.2. *The total cost of one cycle of* $A(\boldsymbol{\tau}^*)$ *is at most* $4(n-1)2^M$.

The desired $8(n-1)$ ratio now follows from:

LEMMA 5.3. *During the time that* $A(\boldsymbol{\tau}^*)$ *completes a cycle, any off-line algorithm B incurs a cost at least* 2^{M-1}.

PROOF. We show that B's transition costs plus the task costs incurred while B is in the same state as $A(\boldsymbol{\tau}^*)$ is at least 2^{M-1}. We proceed by induction on $|E|$, where E defines the MST. For the basis, $|E| = 1$, say $\boldsymbol{\tau}^*$ is $u \to v \to u$. Thus, $d(u, v) \ge 2^{M-1}$. If algorithm B makes an edge traversal, then we are done. Otherwise, B stays at some state (say u). Since $A(\boldsymbol{\tau})$ incurs a cost $d(u,v)$ while in state u, so does B.

For $|E| > 1$, let (u,v) be the edge of maximal weight and MST_1, MST_2 be as in the construction of $\boldsymbol{\tau}^*$.

Case 1. During a cycle of $A(\boldsymbol{\tau}^*)$, B moves from some state in S_1 to some state in S_2 (or vice versa). Then, since MST is a minimum spanning tree, this transition costs at least $d(u,v) \geq 2^{M-1}$.

Case 2. B stays in some S_i (say S_1) during the entire cycle.

(i) $|S_1| = 1$, so $S_1 = \{u\}$. Then, B remains at u throughout the cycle. By the definition of $A(\boldsymbol{\tau})$, a task cost of at least $d(u,v) \geq 2^{M-1}$ is incurred by B while $A(\boldsymbol{\tau}^*)$ is in u.

(ii) $|S_1| > 1$. Let 2^{M_1} be the maximum modified weight in MST_1. Assume first that the traversal $\boldsymbol{\tau}^*$ begins at u or at a vertex in S_2. Then the portion of the cycle spent in S_1 consists of 2^{M-M_1} cycles of $\boldsymbol{\tau}_1$. By the induction hypothesis, B incurs a cost of 2^{M_1-1} in each cycle, for a total cost at least 2^{M-1}. If instead $\boldsymbol{\tau}^*$ begins at a vertex $s \neq u$ in S_1 then the argument is almost the same except that each traversal of $\boldsymbol{\tau}_1$ by $A(\boldsymbol{\tau})$ is interrupted by a visit to S_2, and then resumed later. Consider the moves of B while $A(\boldsymbol{\tau}^*)$ crosses (u,v), completes its traversals of $\boldsymbol{\tau}_2$ and crosses (v,u). Say that during this time B moves from state s' to s''. By the triangle inequality, and since our lower bound for B disregards the task cost, it incurs while $A(\boldsymbol{\tau}^*)$ is in S_2, the relevant cost to B is at least as much as if B waits in state s' until $A(\boldsymbol{\tau}^*)$ returns to u, and then moves to s''. Thus, we can treat this exactly as before. □

6. *An Algorithm with Competitive Ratio* $(2n-1)\psi(d)$

In Section 2, we saw that the cruel taskmaster strategy forces a $2n-1$ competitive ratio in any metrical task system. In this section, we present an on-line algorithm that anticipates the cruel taskmaster strategy and achieves a matching upper bound against that adversary. We then show that the same on-line algorithm guarantees a $2n-1$ ratio for any input sequence of tasks. In the case of asymmetric task systems, a factor of $\psi(d)$ creeps into the analysis.

To motivate the on-line algorithm, assume for now that the adversary follows the cruel taskmaster strategy (for some arbitrarily small ϵ). In this case, each successive task has cost ϵ in the state currently occupied by the on-line algorithm and cost 0 in every other state. Thus, the entire history of the system (including the tasks) depends only on the sequence $s_0, s_1, s_2, \ldots$ of states occupied by the on-line algorithm and the transition times $t_1, t_2, \ldots$ where t_i is the time s_i is entered. Instead of using the t_i's, it is more convenient to parameterize the system history by $c_0, c_1, c_2, \ldots$ where c_k is the cost incurred by the on-line algorithm while in state s_k; here $c_k = (t_{k+1} - t_k)\epsilon$. The choice of $\{s_k\}$ and $\{c_k\}$ constitutes a nearly oblivious on-line algorithm.

How do we choose $\{s_k\}$ and $\{c_k\}$? The key is to understand how the off-line cost changes during some time interval $(\hat{t}, \hat{t} + \Delta t)$.

For a given task sequence $\mathbf{T}$ and an arbitrary time t, let $\phi_t\colon S \to \mathbf{R}$ be the off-line cost function; that is, $\phi_t(s)$ is the minimum cost of processing $\mathbf{T}$ up to time t, given that the system is in state s at time t.

In the present case, where the adversary follows the cruel taskmaster strategy, consider a time $\hat{t}$ at which the on-line algorithm is in state $\hat{s}$. Then, for sufficiently small Δt, $\phi_{\hat{t}+\Delta t}(s) = \phi_{\hat{t}}(s)$ for $s \neq \hat{s}$ since the off-line algorithm can stay in state s during the interval $(\hat{t}, \hat{t} + \Delta t)$. For $s = \hat{s}$, note that the off-line algorithm can either occupy $\hat{s}$ during $(\hat{t}, \hat{t} + \Delta t)$, incurring an additional cost of $\epsilon \Delta t$, or occupy some other state s during $(\hat{t}, \hat{t} + \Delta t)$ and move to $\hat{s}$ at time $\hat{t} + \Delta t$. Thus,

$$\phi_{\hat{t}+\Delta t}(\hat{s}) = \min\left\{\phi_{\hat{t}}(\hat{s}) + \epsilon\Delta t, \min_{s \neq \hat{s}} \phi_{\hat{t}}(s) + d(s, \hat{s})\right\}.$$

This expression indicates that $\phi_t(\hat{s})$ increases with t up until the time $\hat{t}$ such that there is some state s for which $\phi_{\hat{t}}(s) \leq \phi_{\hat{t}}(\hat{s}) - d(s, \hat{s})$. From that time on, the off-line cost remains constant until the next transition by the on-line algorithm. This suggests the following strategy by the on-line algorithm: stay in $\hat{s}$ until the time $\hat{t}$ such that there is some state s such that $\phi_{\hat{t}}(s) \leq \phi_{\hat{t}}(\hat{s}) - d(s, \hat{s})$ and then move to state s. As noted previously, if we assume that the adversary is a cruel taskmaster, then this strategy can be "precomputed" as a nearly oblivious one. We now describe this on-line algorithm explicitly, generalizing it so that it applies to any task sequence.

We define the sequences $\{s_k\}$ and $\{c_k\}$ inductively. At the same time, we also define a sequence $\{f_k\}$ of functions from S to the reals. For intuition, the reader may find it useful to note that in the case that the tasks are given by the cruel taskmaster, $f_k(s)$ is the optimal off-line cost up to the time the on-line algorithm moves into state s_k, subject to the off-line algorithm being in state s at that time. We have:

$$s_0 \quad \text{is given,}$$

$$f_0(s) = 0 \qquad \text{for all} \quad s.$$

For $k \geq 1$, having defined s_{k-1} and f_{k-1}:

$$s_k = \text{state } s \text{ minimizing } f_{k-1}(s) + d(s, s_{k-1})$$

$$f_k(s) = \begin{Bmatrix} f_{k-1}(s) & \text{if} \quad s \neq s_{k-1} \\ f_{k-1}(s_k) + d(s_k, s_{k-1}) & \text{if} \quad s = s_{k-1} \end{Bmatrix}.$$

Having defined the functions $\{f_k\}$ and states $\{s_k\}$, we define for $k \geq 1$,

$$c_{k-1} = f_k(s_{k-1}) - f_{k-1}(s_{k-1}).$$

The nearly oblivious algorithm defined by $\{s_k\}$ and $\{c_k\}$ above is denoted A_d^*. We prove:

THEOREM 6.1. *A_d^* has competitive ratio at most* $(2n - 1)\psi(d)$.

PROOF. Let $\mathbf{T} = T^1 T^2 \cdots T^m$ be any task sequence. Let $t_1 < t_2 < \cdots < t_r \leq m$ be the (continuous) times at which A_d^* prescribes state transitions, that is, the on-line algorithm enters s_k at time t_k. Let $h_k(s)$ be the cost incurred by an optimal off-line algorithm up to time t_k, subject to its being in state s at time t_k. (In terms of the function ϕ defined before, $h_k(s) = \phi_{t_k}(s)$.) Let $h_k = \min_{s \in S} h_k(s)$ be the optimal off-line cost up to time t_k. We want to compare h_k to a_k, the cost of the on-line algorithm up to time t_k. Specifically, to prove Theorem 6.1 it suffices to prove that for some constant L independent of k,

$$h_k \geq \frac{a_k}{(2n - 1)\psi(d)} - L. \tag{6.1}$$

Note first that $a_k = D_k + C_{k-1}$ where

$$D_k = \sum_{i=1}^{k} d(s_{i-1}, s_i)$$

and

$$C_k = \sum_{i=0}^{k} c_i .$$

We need three lemmas:

LEMMA 6.2. *$h_k(s) \geq f_k(s)$ for all $s \in S$.*

LEMMA 6.3. *$f_k(s) - f_k(s') \leq d(s', s)$ for all $s, s' \in S$.*

LEMMA 6.4. *Let*

$$F_k = 2 \sum_{s \neq s_k} f_k(s) + f_k(s_k).$$

Then

$$F_k = C_{k-1} + \sum_{i=1}^{k} d(s_i, s_{i-1}).$$

To deduce (6.1) from the lemmas, we have from Lemma 6.2 and 6.3:

$$\begin{aligned} h_k \geq \min_{s \in S} f_k(s) &\geq \frac{1}{2n-1}[F_k - 2n \max d(i,j)], \\ &\geq \frac{1}{2n-1} F_k - B, \end{aligned}$$

where B is a constant independent of k. Using Lemma 6.4 to substitute for F_k gives

$$h_k \geq \frac{1}{2n-1}\left[C_{k-1} + \sum_{i=1}^{k} d(s_i, s_{i-1})\right] - B. \tag{6.2}$$

Now the distance around the cycle $s_0, s_1, \ldots, s_k, s_0$ is

$$\sum_{i=1}^{k} d(s_i, s_{i-1}) + d(s_0, s_k)$$

and by the definition of $\psi(d)$, this is at least

$$\sum_{i=1}^{k} \frac{d(s_{i-1}, s_i)}{\psi(d)} = \frac{D_k}{\psi(d)}.$$

Using this in (6.2), we get

$$h_k \geq \frac{1}{2n-1}\left[C_{k-1} + \frac{D_k}{\psi(d) - d(s_0, s_k)}\right] - B.$$

Since $\psi(d) \geq 1$, we have $C_{k-1} + D_k/\psi(d) \geq a_k/\psi(d)$. Thus,

$$h_k \geq \frac{a_k}{(2n-1)\psi(d)} - L,$$

where L is independent of k, as required to complete the proof of Theorem 6.1.

It remains to prove the lemmas.

PROOF OF LEMMA 6.2. By induction on k; for $k = 0$ the result is trivial. For $k \geq 1$, we have

$$h_k(s) \geq h_{k-1}(s) \geq f_{k-1}(s) = f_k(s)$$

if $s \neq s_{k-1}$. If $s = s_{k-1}$, we have

$$h_k(s_{k-1}) \geq \min\left\{h_{k-1}(s_{k-1}) + c_{k-1}, \min_{s \neq s_{k-1}} (h_{k-1}(s) + d(s, s_{k-1}))\right\},$$

since either the off-line algorithm is in state s_{k-1} throughout the time interval $[t_{k-1}, t_k)$ or makes a transition from some state s to s_{k-1} during that time interval (in which case the cost incurred prior to that transition is at least $h_{k-1}(s)$). Now

$$h_{k-1}(s_{k-1}) + c_{k-1} \geq f_{k-1}(s_{k-1}) + c_{k-1} = f_k(s_{k-1}),$$

by the induction hypothesis and the definition of c_{k-1}, while

$$\begin{aligned}\min_{s \neq s_{k-1}} (h_{k-1}(s) + d(s, s_{k-1})) &\geq \min_{s \neq s_{k-1}} (f_{k-1}(s) + d(s, s_{k-1}))\\ &= f_{k-1}(s_k) + d(s_k, s_{k-1})\\ &= f_k(s_{k-1})\end{aligned}$$

by the induction hypothesis and the definition of f_k and s_k. □

PROOF OF LEMMA 6.3. By induction on k; the result is obvious for $k = 0$. Suppose it is true for $k - 1$. Then, $f_k(s) - f_k(s') = f_{k-1}(s) - f_{k-1}(s')$ if neither of s, s' equals s_{k-1}. For $s = s_{k-1}$ and $s' \neq s_{k-1}$,

$$\begin{aligned}&f_k(s_{k-1}) - f_k(s')\\ &\quad = f_{k-1}(s_k) + d(s_k, s_{k-1}) - f_k(s')\\ &\quad \leq f_{k-1}(s') + d(s', s_{k-1}) - f_k(s') && \text{(by definition of } s_k)\\ &\quad = d(s', s_{k-1}) && \text{(by definition of } f_k(s')),\end{aligned}$$

and for $s' = s_{k-1}$, $s \neq s_{k-1}$,

$$\begin{aligned}f_k(s) - f_k(s_{k-1}) &= f_{k-1}(s) - f_{k-1}(s_k) - d(s_k, s_{k-1})\\ &\leq d(s_k, s) - d(s_k, s_{k-1}) && \text{(by the induction hypothesis)}\\ &\leq d(s_{k-1}, s). && \square\end{aligned}$$

PROOF OF LEMMA 6.4. By induction on k. For $k = 1$, both sides equal $2d(s_1, s_0)$. For $k \geq 1$, it is enough to show, by induction, that

$$\left(2 \sum_{s \neq s_k} f_k(s) + f_k(s_k)\right) - \left(2 \sum_{s \neq s_{k-1}} f_{k-1}(s) + f_{k-1}(s_{k-1})\right) = c_{k-1} + d(s_k, s_{k-1}).$$

Now, $f_k(s) = f_{k-1}(s)$ if $s \neq s_{k-1}$, so the left-hand side is equal to

$$2f_k(s_{k-1}) - f_{k-1}(s_k) - f_{k-1}(s_{k-1}),$$

which equals $c_{k-1} + d(s_k, s_{k-1})$, by definition. □

7. *Probabilistic On-Line Algorithms*

The result of Section 2 shows that, in a general task system, we cannot hope for a competitive ratio better than $2n - 1$. The ability of the adversary to force this ratio is highly dependent on its knowledge of the current state of the algorithm. This suggests that the performance can be improved by randomizing some of the decisions made by the on-line algorithm. A randomized on-line scheduling algorithm can be defined simply as a probability distribution on the set of all deterministic algorithms. The adversary, knowing this distribution, selects the sequence of tasks to be processed.[2] (Note that if the algorithm is deterministic then the adversary can predict with certainty the response to each successive task.) The resulting schedule $\boldsymbol{\sigma}$ is a random variable and (using the notation of Section 1) we are interested in the relationship between $\bar{c}_A(\mathbf{T}) = \sum_{\sigma} c(\mathbf{T}; \boldsymbol{\sigma}) \cdot \mathrm{pr}(\boldsymbol{\sigma} | \mathbf{T})$ as it relates to $c_0(\mathbf{T})$, where $\mathrm{pr}(\boldsymbol{\sigma} | \mathbf{T})$ denotes the probability that A follows $\boldsymbol{\sigma}$ on input $\mathbf{T}$. The expected competitive ratio of A, $\bar{w}(A)$, is the infimum over all w such that $\bar{c}_A(\mathbf{T}) - wc_0(\mathbf{T})$ is bounded above for any finite task sequence, and the randomized competitive ratio, $\bar{w}(s, d)$, of the task system (S, d) is the infimum of $\bar{w}(A)$ over all randomized on-line algorithms A.

The above definition can be described in terms of the scheduler-taskmaster game described in Section 2. The ith state $\boldsymbol{\sigma}(i)$ is a function of the first i tasks $T^1, \ldots, T^i$, the first i states $\boldsymbol{\sigma}(0), \boldsymbol{\sigma}(1), \ldots, \boldsymbol{\sigma}(i-1)$, and some random bits. For a given sequence of i tasks, the schedule followed up to that time is a random variable and the adversary selects the $(i+1)$st task knowing the distribution of this random variable, but not the actual schedule that was followed. It is this uncertainty that gives randomized algorithms a potential advantage over purely deterministic ones.

Thus, far we have been unable to analyze the expected competitive ratio for arbitrary task systems. However, for the uniform task system, the system where all state transitions have identical (say unit) cost, we have been able to determine the expected competitive ratio to within a factor of 2. In this case, randomization provides a substantial reduction in the competitive ratio. (Recall that the competitive ratio for a deterministic algorithm is at least $2n - 1$.) Let $H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$. It is well known that $H(n)$ is between $\ln n$ and $1 + \ln n$.

[2] Such an adversary is now called an "oblivious adversary". Adaptive adversaries were introduced in Raghavan and Snir [18] and further studied in Ben-David et al. [1].

THEOREM 7.1. *If (S, d) is the uniform task system on n states, then*

$$H(n) \leq \overline{w}(S, d) \leq 2H(n).$$

PROOF OF THE UPPER BOUND. Note first that Lemma 3.1 extends to randomized on-line algorithms and thus it suffices to describe a continuous-time algorithm that achieves the desired expected competitive ratio. The algorithm proceeds in a sequence of phases. The time at which the ith phase begins is denoted t_{i-1}. For a state s, let A_s be the algorithm that stays in state s for the entire algorithm. A state s is said to be *saturated for phase i at time* $t > t_{i-1}$, if the cost incurred by A_s during the time interval $[t_{i-1}, t]$ is at least 1. The ith phase ends at the instant t_i when all states are saturated for that phase. During a phase the on-line algorithm proceeds as follows: remain in the same state until that state becomes saturated and then jump randomly to an unsaturated state. Note that at the end of a phase, the on-line algorithm moves with equal probability to some state, with all states again becoming unsaturated.

To analyze the expected competitive ratio of this algorithm during a single phase, any off-line algorithm incurs a cost of at least 1, either because it makes a state transition or because it remains in some state during the entire phase and that state becomes saturated. Hence, it suffices to show that the expected cost of the on-line algorithm is at most $2H(n)$. Let $f(k)$ be the expected number of state transitions until the end of the phase given that there are k unsaturated states remaining. Note $f(1) = 1$. For $k > 1$, since the algorithm is in each unsaturated state with equal probability, the probability that the next state to become saturated results in a transition is $1/k$. Thus, $f(k) = f(k-1) + 1/k$, so that $f(k) = H(k)$ and the expected cost of the phase (task processing costs + transition costs) is at most $2H(n)$. □

PROOF OF THE LOWER BOUND. Yao [23] has noted that using the von Neumann minimax theorem for two-person games, a lower bound on the performance of randomized algorithms in these (and most other) models can be obtained by choosing any probability distribution on the input (in this case, the task sequence) and proving a lower bound on the expected cost of any deterministic algorithm on this input. We use this principle to derive a lower bound on the expected competitive ratio of a randomized algorithm on a uniform task system.

If $\mathbf{T}$ is an infinite task sequence, let $\mathbf{T}^j$ denote the first j tasks.

LEMMA 7.2. *Let (S, d) be a task system and D be a probability measure on the set of infinite task sequences. Let E denote expectation with respect to D and suppose that $E(c_0(\mathbf{T}^j))$ tends to infinity with j. Let m_j denote the minimum over all deterministic on-line algorithms A of $E(c_A(\mathbf{T}^j))$. Then*

$$\overline{w}(S, d) \geq \limsup_{j \to \infty} \frac{m_j}{E(c_0(\mathbf{T}^j))}.$$

PROOF. By the definition of $\overline{w}(S, d)$, there exists a randomized algorithm R and a constant c such that for all infinite task sequences $\mathbf{T}$ and integers j, $c_R(\mathbf{T}^j) - \overline{w}(S, d)c_0(\mathbf{T}^j) < c$. Averaging over the distribution D, we obtain that for each j, $E(c_R(\mathbf{T}^j)) - \overline{w}(S, d)E(c_0(\mathbf{T}^j)) < c$ or $\overline{w}(S, d) > E(c_R(\mathbf{T}^j))/E(c_0(\mathbf{T}^j)) - c/E(c_0(\mathbf{T}^j))$. Since $E(c_R(\mathbf{T}^j)) \geq m_j$ and $E(c_0(\mathbf{T}^j))$ tends to infinity we obtain $\limsup m_j/E(c_0(\mathbf{T}^j)) \leq \overline{w}(S, d)$. □

Now for $s \in S$, define U_s to be the unit elementary task with a processing cost of 1 in state s and 0 in any other state. Let D be the distribution on task sequences obtained by generating each successive task independently and uniformly from the unit elementary tasks. We show that for each j, $m_j \geq j/n$ and $E(c_0(\mathrm{T}^j)) \leq j/(nH(n)) + O(1)$. Applying Lemma 7.2, we obtain

$$\overline{w}(S,d) \geq \limsup_{j\to\infty} \frac{j/n}{j/(nH(n) + O(1))} = H(n),$$

as required.

Identify a sequence of unit elementary tasks with the sequence of states $s_1, s_2, \ldots,$ that define each task. Note that if $\boldsymbol{\sigma}$ is the schedule produced by a deterministic on-line algorithm A, then, for each time step j, A incurs a cost of 1 at time step j if $s_j = \boldsymbol{\sigma}(j-1)$. This happens with probability $1/n$, and hence the expected cost of processing each task is at least $1/n$ and $m_j \geq j/n$.

Now for each time i, let $a(i)$ be the least index greater than i such that the sequence $s_{i+1}, s_{i+2}, \ldots, s_{a(i)}$ contains all states of S. Consider the following off-line algorithm: letting U_{s_i} denote the ith task, at time i do not change state unless $\boldsymbol{\sigma}(i-1) = s_i$, in which case let $\boldsymbol{\sigma}(i) = s_{a(i)}$. (This is in fact the optimal off-line strategy for each such task sequence, although we do not need that fact.) We claim that the expected cost (with respect to the distribution D) of this strategy after the first j tasks, as a function of j, is $j/(nH(n)) + O(1)$. The algorithm incurs a cost of one exactly at those times i such that $\boldsymbol{\sigma}(i-1) = s_i$, and incurs zero cost at other times. Let c_j be the cost incurred up to time j and let X_k be the time at which the total cost incurred by the algorithm reaches k. Then, $c_j = \max\{k\colon X_k \leq j\}$ and the quantities: $Y_k = X_k - X_{k-1}$ are independent identically distributed random variables, which means that the sequence $\{c_j\colon j \geq 1\}$ is a renewal process. By the elementary renewal theorem (see [20, Theorem 3.8]), the limit of the ratio $\mathrm{Exp}(c_j)/j$ exists and is $1/\mathrm{Exp}(Y_k)$. To compute $\mathrm{Exp}(Y_k)$, note that Y_k is the time until a sequence of randomly generated states contains all of the states. When i of the states have appeared in the sequence the probability that the next state is a new state is $(n-i)/n$ and hence the expected number of states between the ith and $(i+1)$st state has appeared is $n/(n-i)$. Summing on i yields $\mathrm{Exp}(Y_k) = nH(n)$. □

Upper and lower bounds on the expected competitive ratio for other specific task systems would be of interest. In particular, we would like to know what it is for the system in which d is the graph distance on a cycle; that is, $d(s_i, s_j) = \min(|j-i|, n - |j-i|)$. We suspect that there is an $O(\log n)$ upper bound for any task system on n states, but the analysis of $\overline{w}(S,d)$ for arbitrary task systems seems to be a challenging problem.

8. *Conclusion*

We have introduced an abstract framework for studying the efficiency of on-line algorithms relative to optimal off-line algorithms. With regard to deterministic algorithms, we have established an optimal $2n-1$ bound on the competitive ratio for any n state metrical task system. For randomized algorithms, we have only succeeded in analyzing the expected competitive ratio for the uniform task system.

One must ask about the "utility" of such an abstract model. (For the utility of an even more abstract model, we refer the reader to Ben-David et al. [1]). As stated in the introduction, our optimal bound for deterministic algorithms is unduly pessimistic in that our model allows arbitrary tasks. We believe that the utility of a general model should not only be measured in terms of direct application of results, but one should also consider the extent to which the model, techniques and results suggest results in more concrete settings.

One such very appealing setting is the k-server model introduced by Manasse et al. [16]. Here, k mobile servers satisfy requests that come in the form of a command to place a server on one of n nodes in a graph G whose edges have lengths that satisfy the triangle inequality. (In their definition, the edge lengths need not be symmetric but since all the results in [16] assume symmetry, we make this assumption to simplify the discussion.) In fact, the model permits n to be infinite as, for example, when one takes G to be all of R^d. As observed in [16], there is a strong relation between the $n = k + 1$ node, k server problem and task systems. To be precise, an $n = k + 1$ node, k server problem is equivalent to an n state task system where the tasks are restricted to be ∞-elementary tasks (i.e., those that have a cost vector that is zero in $n - 1$ states and ∞ in that remaining state). A variant of the k server problem called the k-server with excursions problem (where servers can either move to service a request or else make an excursion where the costs of such excursions are defined by a given matrix) is also introduced and in this case the $n = k + 1$ node, k server with excursion problem is equivalent to task systems where the tasks are restricted to arbitrary elementary tasks. For both problems, one studies the *competitive ratio*. Our $2n - 1$ upper bound then immediately applies in the $n = k + 1, k$ server (with or without excursions) case but the lower bound only applies directly to the k server with excursion problem. Indeed Manasse et al. [16] show that $n - 1$ is the optimal bound for the competitive ratio of a $n = k + 1$ node, k server problem. Their $n - 1$ lower bound is derived by using the same adversary (i.e., the cruel taskmaster) as in our Theorem 2.2 and then applying a nice averaging argument. (Alternatively, the proof of Theorem 2.2 could be modified so as to yield the optimal $n - 1$ lower bound when tasks are restricted to ∞-elementary tasks.) The $k = n - 1$ upper bound requires a new and intuitively appealing algorithm. By a quite different algorithm, Manasse et al. [16] also prove that for all $k = 2$ server problems, the competitive ratio is 2 and they make the following intriguing conjecture: the competitive ratio for every k-server problem (without excursions) is k. A lower bound of k is immediate from the previous discussion since such a bound can be derived when one restricts the adversary to $n = k + 1$ nodes. Our task system framework provides a little positive reinforcement for the conjecture in that our deterministic upper bound can be used to show that the competitive ratio for any n-node, k-server problem is at most $2\binom{n}{k} - 1$ which (at least) is independent of the edge lengths. Here each of the $\binom{n}{k}$ possible positions of the servers becomes a state of the task system and the induced state transition costs inherit the triangle inequality and symmetry properties. A node request becomes a task whose components are either zero (if the state contains the requested node) or infinite (if the state does not contain the requested node). Although a state transition may represent moving more than one server, as noted in [16], we need only move the server that covers the request and treat the state as representing the virtual position of servers while remembering the physical position of all servers. By the same simulation, our Theorem 6.1 also provides some information for nonsymmetric k-server problems. We note that there is now an impressive literature concerning the k-server conjecture, including optimal results on specific server systems (e.g., [5], [6]), as well as results yielding upper bounds (exponential in k) for arbitrary k-server systems (see [10] and [12]).

We note however that the k-server with excursion problem cannot, in general, be formulated as a task system problem. For now, if a state represents the position of the servers, the cost of a task is not just a function of the state but also of how we reached this state. Indeed, for $n \geq k + 2$, Shai Ben-David (personal communication) has shown that the competitive ratio can be made to grow with the edge lengths. (However, we should note that Ben-David's example uses excursion costs which do not satisfy the triangle inequality.) One suggestion is to redefine the problem so that servers can "move and serve" rather than "move or serve". This point of view is investigated in the case of one server by Chung et al. [7].

Our results on the expected competitive ratio of randomized algorithms for the unit-cost task system has a very nice analogue in the unit-cost k-server problem that is better known as the paging or caching problem. For the paging problem, Fiat et al. [9] have established a $2H(k)$ upper bound on the expected competitive ratio using a very appealing (and possibly quite practical) randomized algorithm. It is interesting to note that in the (admittedly nonpractical) case when $n = k + 1$, their algorithm becomes the algorithm of our Theorem 7.1. A direct modification of our lower bound in Theorem 7.1 can be used to establish an $H(k)$ lower bound for the expected competitive ratio of randomized on-line paging algorithms. Subsequently, McGeoch and Sleator [17] have constructed a randomized algorithm with a matching upper bound of $H(k)$. A number of other important results on paging problems, on the trade-off between randomization and memory, and on a more general study of randomized algorithms in the context of competitive ratios for on-line algorithms (against various types of adversaries) can be found in Raghavan and Snir [18], Coppersmith et al. [8], and Ben-David et al. [1].

The papers of Manasse et al. [19], Fiat et al. [9], and Raghavan and Snir [18] and the original papers by Sleator and Tarjan [21] and Karlin et al. [15] argue for the development of a theory (or rather theories) for competitive analysis. Such a study seems fundamental to many other disciplines (e.g., economics) in addition to computer science and, as such, we expect to see a number of alternative models and performance measures.

ACKNOWLEDGMENTS. The idea of developing a general model for on-line task systems was introduced to us in various conversations with Fan Chung, Larry Rudolph, Danny Sleator, and Marc Snir. Don Coppersmith suggested that randomization may be useful in this context. We gratefully acknowledge their contributions to this paper. Finally, we thank the referees and Gordon Turpin for their careful reading and suggestions concerning the submitted manuscript.

REFERENCES

1. BEN-DAVID, S., BORODIN, A., KARP, R., TARDOS, G., AND WIGDERSON, A. On the power of randomization in online algorithms. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (Baltimore, Md., May 12–14). ACM, New York, 1990, pp. 379–386.

2. BENTLEY, J. L., AND MCGEOCH, C. C. Amortized analyses of self-organizing sequential search heuristics. *Commun. ACM 28*, 4 (Apr. 1985), 404–411.
3. BITNER, J. R. Heuristics that dynamically alter data structures to reduce their access time. Ph.D. dissertation. Univ. Illinois, 1976.
4. BITNER, J. R. Heuristics that dynamically organize data structures. *SIAM J. Comput. 8* (1979), 82–110.
5. CHROBAK, M., KARLOFF, H. J., PAYNE, T., AND VISHWANATHAN, S. New results on server problems. *SIAM J. Disc. Math. 4*, 2 (1991), 172–181.
6. CHROBAK, M., AND LARMORE, L. An optimal on-line algorithm for the server problem on trees. *SIAM J. Comput. 20* (1991), 144–148.
7. CHUNG, F. R. K., GRAHAM, R. L., AND SAKS, M. A dynamic location problem for graphs. *Combinatorica 9* (1989), 111–131.
8. COPPERSMITH, D., DOYLE, P., RAGHAVAN, P., AND SNIR, M. Random walks on weighted graphs and applications to on-line algorithms. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing* (Baltimore, Md., May 12–14). ACM, New York, 1990, pp. 369–378.
9. FIAT, A., KARP, R., LUBY, M., MCGEOCH, L., SLEATOR, D., AND YOUNG, N. Competitive paging algorithms. *J. Algorithms 12* (1991), 685–699.
10. FIAT, A., RABANI, Y., AND RAVID, Y. Competitive k-Server Algorithms. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. IEEE, New York, 1990, pp. 454–463.
11. GONNET, G., MUNRO, J. I., AND SUWANDA, H. Exegesis of self-organizing linear search. *SIAM J. Comput. 10*, 3 (1981), 613–637.
12. GROVE, E. The harmonic online k-server algorithm is competitive. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (New Orleans, La., May 6–8). ACM, New York, 1991, pp. 260–266.
13. HEYMAN, D., AND SOBEL, M. *Stochastic Models in Operations Research*. vol. II. McGraw-Hill, New York, 1984.
14. KAN, Y. C., AND ROSS, S. M. Optimal list order under partial memory constraints. *J. Appl. Prob. 17* (1980), 1004–1015.
15. KARLIN, A. R., MANASSE, M. S., RUDOLPH, L., AND SLEATOR, D. D. Competitive snoopy caching. *Algorithmica 3*, 1 (1988), 79–119.
16. MANASSE, M., MCGEOCH, L., AND SLEATOR, D. Competitive algorithms for server problems. *J. Algorithms 11* (1990), 208–230.
17. MCGEOCH, L. A., AND SLEATOR, D. D. A strongly competitive randomized paging algorithm. *Algorithmica*, to appear.
18. RAGHAVAN, P., AND SNIR, M. "Memory versus randomization in on-line algorithms", Automata, Languages and Programming, Lecture Notes in Computer Science v. 372, Springer-Verlag, New York, 1989, pp. 687–703.
19. RIVEST, R. On self-organizing sequential search heuristics. *Commun. ACM 19*, 2 (Feb. 1976), 63–67.
20. ROSS, S. *Applied Probability Models with Optimization Applications*. Holden-Day, Calif. 1970.
21. SLEATOR, D. D., AND TARJAN, R. E. Amortized efficiency of list Update and Paging rules. *Commun. ACM 28* (1985), 202–208.
22. TARJAN, R. E. Amortized computational complexity. *SIAM J. Alg. Disc. Math. 6* (1985), 306–318.
23. YAO, A. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. IEEE, New York, 1977, pp. 222–227.

RECEIVED JULY 1988; REVISED APRIL 1991; ACCEPTED MAY 1991