Unfair Problems and Randomized Algorithms

Steve Seiden*

Institut für Mathematik B, Technische Universität, Graz,
Steyrergasse 30/2 Stock, A-8010 Graz, Austria
E-mail: sseiden@acm.org

Borodin, Linial, and Saks introduced a general model for online systems called metrical task systems (1992, J. Assoc. Comput. Mach. 39(4), 745–763). In this paper, the unfair two state problem, a natural generalization of the two state metrical task system problem, is studied. A randomized algorithm for this problem is presented, and it is shown that this algorithm is optimal. Using the analysis of the unfair two state problem, a proof of a decomposition theorem similar to that of Blum, Karloff, Rabani, and Saks (1992, “Proc. 33rd Symposium on Foundations of Computer Science,” pp. 197–207) is presented. This theorem allows one to design divide and conquer algorithms for specific metrical task systems. Our theorem gives the same bounds asymptotically, but it has less restrictive boundary conditions.

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 (Karlin et al. 1994, Manasse et al. 1990, Sleator and Tarjan 1985b, Westbrook 1992). 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, 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 et al. introduced task systems in (Borodin et al. 1992) 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

* This research was done while the author was a graduate student at the Department of Information and Computer Science of the University of California, Irvine.
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 \(S = \{s_1, ..., s_n\}\) is a set of \(n\) states and \(d = (d(s, t))\), the distance matrix, is an \(n \times n\) nonnegative 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, \(\sigma = T_1, T_2, ..., T_l\).

A task is a vector with \(n\) nonnegative 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 \(T(s) > 0\).

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 \(\pi\), a function from \(\{0, 1, ..., l\}\) 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 \(T_1, ..., T_i\). The cost of a schedule \(\pi\) on \(\sigma\) is the sum of the cost of moving from state to state (the moving cost) and the cost of processing tasks (the stationary cost)

\[
\text{cost}(\pi, \sigma) = \sum_{i=1}^{l} d(\pi(i-1), \pi(i)) + \sum_{i=1}^{l} T_i(\pi(i)).
\]

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

\[
\text{cost}_{opt}(\sigma) = \min_{\pi} \text{cost}(\pi, \sigma).
\]

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 \(\mathcal{A}\) is said to be \(c\)-competitive if there is a constant \(a\) such that for all \(\sigma\),

\[
\text{cost}_{\mathcal{A}}(\sigma) \leq c \cdot \text{cost}_{opt}(\sigma) + a.
\]

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

\[
E[\text{cost}_{\mathcal{A}}(\sigma)] \leq c \cdot \text{cost}_{opt}(\sigma) + a.
\]
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 (Sleator and Tarjan 1985a). The term competitive analysis originated in (Karlin et al. 1988). 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 (Ben-David et al. 1994). We utilize an oblivious adversary, which is an adversary that does not know the random choices of the algorithm before determining the input sequence.

In the case of deterministic algorithms, Borodin et al. show in (Borodin et al. 1992) 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. This has lead researchers to consider algorithms for 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 \((d(s,t) = 1 \text{ for all } s \neq t)\). We call such task systems uniform task systems. For uniform task systems, Irani and Seiden (1995) improve the upper bound of \( 2H_n \) to

\[
\frac{H_n}{\ln 2} + 1 \approx 1.4427H_n + 1.
\]

In more recent work, these same authors (Irani and Seiden 1998) improve this bound to

\[
H_n \cdot O(\sqrt{\log n}).
\]

For the metric space consisting of \( n \) evenly spaced points on the line, Blum et al. (1997) present an algorithm for the \((n - 1)\)-server problem which is

\[
2^{\Omega(1) \sqrt{\log\log\log n}}
\]

competitive. This function is sublinear, but superlogarithmic. This algorithm is easily adapted to metrical task systems. A star space is a metric space where there is center point \( c \), and the distance \( d(s,t) \) between any two points \( s \neq c \) and \( t \neq c \) is \( d(s,c) + d(c,t) \). Blum, Furst, and Tomkins show an \( O(\log^2 n) \) competitive algorithm for any such space. Chrobak and Noga show that the randomized competitive ratio for two states is exactly 2. Previously, the best known upper bound applicable to all spaces was

\[
\left( \frac{e}{e - 1} \right)^n \frac{1}{e - 1},
\]
which is from Irani and Seiden (1995, 1998). Recently, several asymptotic results have been shown. Bartal (1996) shows a

$$2O\left(\sqrt{\log(n + d)} \log \log n\right)$$

competitive algorithm for all spaces with diameter $d$. This is sublinear when the diameter is polynomial in $n$. Bartal et al. (1997) improve this by giving an $O(\log^6 n)$-competitive algorithm for general spaces.

The best lower bound applicable to all spaces is

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

from Blum et al. (1992). This result follows from a theorem, proved in (Blum et al. 1992), called the decomposition theorem. Suppose we have a metric space which consists of two widely separated subspaces. We call such metric spaces decomposable. We shall define this formally in Section 3. Given upper and lower bounds on the competitive ratios of the subspaces, we wish to find upper and lower bounds on the competitive ratio of the entire space. The decomposition theorem allows us to accomplish this.

We consider a simple problem which we call the unfair two state problem, derive an algorithm called Two Stable for solving it, and show that this algorithm is optimal by giving a matching lower bound. The unfair two state problem is an interesting problem in and of itself. However, our main interest in this problem stems from the fact that Two Stable can be used to design divide and conquer randomized algorithms for decomposable metric spaces. This algorithm has recently been independently investigated by Bartal et al. (1997), who use it as a subroutine in an $O(\text{polylog}(n))$-competitive randomized algorithm for metrical task systems. Thus the analysis of the unfair two state problem plays a key role in the resolution of the randomized metrical task system problem. In this paper, we use the analysis of the unfair two state problem to derive a decomposition theorem similar to that of Blum et al. (1992). This theorem allows one to design divide and conquer algorithms for specific metrical task systems. Our theorem gives the same bounds asymptotically, but has less restrictive boundary conditions. Further, our proof makes use of work functions, which have become standard in the design and analysis of online algorithms.

The proof of the decomposition theorem presented in (Blum et al. 1992) is proved by analyzing the walker jumper game. The idea of the proof is to show a correspondence between a player of the game and a randomized algorithm in a decomposable metric space. The idea of the proof presented here is to show a correspondence between the unfair algorithm for the two state problem and a randomized algorithm in a decomposable metric space. Because the unfair two state problem is more general than the walker jumper game, we get a tighter

1 The walker jumper game has no parameter corresponding to what we call $\gamma$. 
correspondence, and thus the boundary conditions on the theorem presented here are less restrictive.

In Section 2 we study the unfair two state problem. In Section 3 we prove the decomposition theorem. In Section 4, we mention some applications of the theorem.

2. THE UNFAIR TWO STATE PROBLEM

In the two state metrical task system problem, we have two states u and v. Without loss of generality, the distance between them is 1.

We generalize the two state problem as follows. Consider a task T which charges x for being in state u and y for being in v. The problem is "unfair" in the following sense: the adversary is charged x for being in state u, while the algorithm is charged x - x. Similarly, the adversary is charged y for being in state v, while the algorithm is charged β - y. Finally, the adversary is charged d(u, v) = 1 for moving between u and v, while the algorithm is charged γ. We assume that x ≥ β ≥ 1, and that γ > 0.

The unfair two state problem models the following situation: We have two algorithms which achieve competitive ratios of x and β, respectively, on the two subspaces of a decomposable space. Without loss of generality, the minimum distance between the two subspaces is 1. The maximum distance is γ. If we have an algorithm for the unfair two state problem, we could combine it with the two subspace algorithms to get an algorithm for the entire space.

Throughout this section, u and v refer to the two states of our task system.

2.1. Preliminaries

Let ℝ* be the set of nonnegative real numbers.

For convenience, we adopt a continuous time model also used in (Borodin et al. 1992), 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, ℓ) to S such that for each state s, π−1(s) is a finite disjoint union of half open intervals [ti, ti+1). In addition, we require that π(0) = s1. There are a finite number of transition times t1 < t2 < ... < tk. Denote the state to which the algorithm moves at time ti by xi = π(ti). We define t0 = 0. The cost for the schedule is then

\[ \text{cost}(\pi, σ) = \sum_{i=1}^{k} d(x_{i-1}, x_i) + \sum_{i=1}^{ℓ} \int_{j_{i-1}}^{j_i} T_i(π(t)) \, dt. \]

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.

The work function is a function w: S → ℝ* where w(u) is the optimal offline cost to process the first i tasks and end up in state u. The idea of work functions was first introduced in (Borodin et al. 1992). The actual term work function was coined
in (Chrobak and Larmore 1991). The work function for metrical task systems is defined as

\[
    w_0(u) = d(u, s_1)
\]

\[
    w_{i+1}(u) = \min_{v \in S} \{ w_i(v) + T_{i+1}(v) + d(u, v) \}.
\]

When we omit the subscript \( i \) we are referring to the work function at the current point in the task sequence. Note that \( \min_u w(u) \) is exactly the optimal offline cost. For any work function \( w \) and any two states \( u \) and \( v \),

\[
    |w(u) - w(v)| \leq d(u, v).
\]

This is known as the slope condition. We say that state \( u \) is dominated by state \( v \) if

\[
    w(u) = w(v) + d(u, v).
\]

If there exists a state \( v \neq u \) such that \( u \) is dominated by \( v \) then we say that \( u \) is dominated. We extend the definition of the work function to the continuous time interval. Let \( i \) be an integer in \( \{0, \ldots, l-1\} \) and \( \lambda \) be a real number such that \( 0 < \lambda \leq 1 \). The work function at time \( t = i+\lambda \) is

\[
    w_i(u) = \min_{v} \{ w_i(v) + \lambda T_{i+1}(v) + d(u, v) \}.
\]

Note that for any positive integer \( i \), \( w_i(s) \) is the same under the extended definition. Also note that \( w_i(u) \) is a continuous function of \( t \) for \( t \in [0, l] \) and for all states \( u \). When we drop the \( t \) subscript, we are referring to the value of \( w \) at the current point in time.

We define the useful work of a task \( T_i \) to be the amount by which \( T_i \) increases the work function. Formally, the useful work \( U_i \) of \( T_i \) is

\[
    U_i = \sum_{u \in S} (w_i(u) - w_{i-1}(u)).
\]

We make use of the following well known lemma:

**Lemma 2.2.** For a sequence of \( l \) tasks let \( U_i \) be the useful work of task \( T_i \) and let \( U^* = \sum_{i=1}^{l} U_i \). Then the optimal cost is at least

\[
    \frac{U^*}{n} - d_{\text{max}},
\]

where \( d_{\text{max}} = \max_{u,v \in S} d(u, v) \).

Applied to the unfair two state problem, this lemma allows us to charge the adversary \( x/2 \) whenever the work function of the some state increases by \( x \).
2.2. The Two Stable Algorithm

We design a stable randomized algorithm for the unfair two state problem. A stable algorithm is one whose probability distribution is a function solely of the current work function (Chrobak and Larmore 1991, Lund and Reingold 1994). The algorithm is a generalization of Chrobak and Noga’s algorithm for the fair two state problem. The algorithm defined is a continuous time one. The probability distribution of our algorithm is determined by

$$\hat{w} = w(u) - w(v).$$

Note that $-1 \leq \hat{w} \leq 1$.

The Two Stable algorithm is defined as follows: If $\hat{w} = x$ then the probability that the algorithm is in $u$ is $p(x)$. (We derive this distribution later.) The probability that the algorithm is in $v$, of course, $1 - p(x)$. We set $p(1) = 0$ and $p(-1) = 1$. We assume that $p(x)$ is a “smooth” function which is monotone nonincreasing in $x$.

We divide the time line into periods. A period is a continuous time interval, during which one of three conditions holds:

1. $\hat{w}$ is monotone increasing.
2. $\hat{w}$ is monotone decreasing.
3. $\hat{w}$ does not change.

It is easily seen that the entire time line may be covered by periods. Case 3 may be immediately disposed of; if $\hat{w}$ does not change, then both states are being charged at the same rate. The algorithm does not move, and any cost incurred by the algorithm is matched by adversary cost.

We define the potential to be

$$\phi = -c \frac{x}{2} - \gamma p(x) + \beta \int_{-1}^{x} (1 - p(z)) \, dz,$$

where $c$ is the competitive ratio of the algorithm. Since $p$ is nonnegative, we have

$$-c \frac{x}{2} - \gamma \leq \phi \leq c \frac{x}{2} + \beta,$$

for all $-1 \leq x \leq 1$.

The intuition behind our potential function is as follows: We assume that the final work function is equal to the initial one; making this assumption only adds a finite cost to the sequence. Given this fact, we know that if $\hat{w}$ increases from $x$ to $x'$, it must eventually decrease from $x'$ to $x$. The potential function pairs these two events.

Consider first Case 1. The value of $\hat{w}$ is positive and it increases in a monotone fashion from $x$ to $x'$. The probability that the algorithm is in $u$ decreases from $p(x)$ to $p(x')$. The cost to the algorithm is $c \frac{x - x'}{2}$.
to \( p(x') \). The algorithm therefore moves at a cost of \( \gamma \) with probability \( p(x) - p(x') \). The algorithm also incurs a stationary cost of

\[
\alpha \int_x^{x'} p(z) \, dz
\]
as \( w \) changes from \( x \) to \( x' \). The total is

\[
\alpha \int_x^{x'} p(z) \, dz + \gamma (p(x) - p(x')).
\] (1)

When \( w \) changes from \( x \) to \( x' \), the change in potential is

\[
\Delta \phi = -c \frac{x' - x}{2} - \gamma p(x') + \beta \left[ \gamma (1 - p(z)) \right]_{-1}^{x'}
\]

\[
= -c \frac{x' - x}{2} + \gamma (p(x) - p(x')) + \beta \int_x^{x'} (1 - p(z)) \, dz.
\]

By Lemma 2.2, we charge the adversary \( (x' - x)/2 \). To ensure that the algorithm is \( c \)-competitive, we derive a \( p \) such that for all \( x \) and \( x' > x \)

\[
c \frac{x' - x}{2} = \gamma (p(x) - p(x')) + \alpha \int_x^{x'} p(z) \, dz + \Delta \phi
\]

\[
= -c \frac{x' - x}{2} + 2\gamma (p(x) - p(x')) + \alpha \int_x^{x'} p(z) \, dz + \beta \int_x^{x'} (1 - p(z)) \, dz
\]

\[
= -c \frac{x' - x}{2} + 2\gamma (p(x) - p(x')) + \beta (x' - x) + (\alpha - \beta) \int_x^{x'} p(z) \, dz.
\]

This is true if and only if

\[
(c - \beta)(x' - x) = 2\gamma (p(x) - p(x')) + (\alpha - \beta) \int_x^{x'} p(z) \, dz.
\]

We rewrite the above as

\[
g(x' - x) = h(P'(x) - P'(x')) + f(P(x') - P(x)),
\]
where
\[ p(x) = P'(x) = \frac{dP(x)}{dx}, \]
\[ f = \alpha - \beta, \]
\[ g = \epsilon - \beta, \]
\[ h = 2\gamma. \]

Separating variables we find that
\[ gx' + hP'(x') - fP(x') = gx + hP(x) - fP(x). \]

In other words:
\[ gx + hP(x) - fP(x) = a, \]
for all \( x \), for some constant \( a \). We first consider the case where \( x \neq \beta \) and therefore \( f \neq 0 \). Solving the above differential equation, we find that
\[ p(x) = \frac{g}{f} + b \frac{fe^{f(x)b}}{h}, \]
for some constant \( b \). We know that \( p(1) = 0 \) and \( p(-1) = 1 \) and therefore we derive
\[ b = \frac{2e^{(x + \beta)(2\gamma)}}{(\beta - \alpha)(e^{\gamma} - e^{\beta \gamma})}, \quad c = \frac{xe^{\beta \gamma} - xe^{\gamma}}{e^{\gamma} - e^{\beta \gamma}}. \]

Note that, as one would expect, the value of \( c \) is symmetric in \( \alpha \) and \( \beta \). Substituting and simplifying we find that
\[ p(x) = \frac{e^{x \gamma}(e^{(x + \beta)(2\gamma)} - 1)}{e^{\beta \gamma} - e^{\gamma}}. \]

By similar methods, we derive that
\[ p(x) = \frac{1 - x}{2} \]
and \( c = \alpha + \gamma \) when \( \alpha = \beta \).

We now consider Case 2, where \( \nu \) is charged, causing \( \nu' \) to decrease to some value \( x' \). The cost incurred by the algorithm is
\[
\gamma((1 - p(x')) - (1 - p(x))) + \beta \int_{x'}^{x} (1 - p(z)) \, dz
\]
\[ = \gamma(p(x') - p(x)) + \beta \int_{x'}^{x} (1 - p(z)) \, dz. \quad (2) \]
The change in potential is
\[
A\phi = -c \frac{x' - x}{2} + \gamma (p(x) - p(x')) + \beta \int_x^{x'} (1 - p(z)) \, dz. \tag{3}
\]

The sum of (2) and (3) is just \(c(x - x')/2\). Note that the adversary's cost is \((x - x')/2\), and we are done with this case.

We define
\[
c = \begin{cases} 
\alpha + \gamma & \text{if } \alpha = \beta \\
\frac{\beta e^{\alpha/\gamma} - \beta e^{\gamma/\beta}}{e^{\alpha/\gamma} - e^{\gamma/\beta}} & \text{otherwise}. 
\end{cases} \tag{4}
\]

Let \(\phi_{max} = \max_{x} |\phi|\). We know that \(-c/2 - \gamma \leq \phi \leq c/2 + \beta\). If \(\phi\) is nonnegative then \(|\phi| = \phi \leq c/2 + \beta \leq 3c/2\). If \(\phi\) negative then \(|\phi| \leq c/2 + \gamma \leq 3c/2\). Therefore \(\phi_{max} \leq 3c/2\).

Let \(w\) and \(\phi\) be the work function and the potential function at the end of the request sequence, respectively. Let \(w_0\) and \(\phi_0\) be the initial work function and the initial potential function. We have shown that the algorithm incurs an expected cost of at most

\[
c \frac{w(u) - w_0(u) + w(v) - w_0(v)}{2} + \phi - \phi_0 \
\leq c \frac{w(u) + w(v)}{2} + 2\phi_{max} \\
\leq c \cdot \min\{w(u), w(v)\} + \frac{7c}{2} \\
\leq c \cdot \text{cost}_{opt}(\sigma) + \frac{7c}{2},
\]

over any request sequence. Since the optimal offline cost is \(\min\{w(u), w(v)\}\), the algorithm is \(c\) competitive.

2.3. A Lower Bound for the Unfair Two State Problem

In this section we show a matching lower bound.

Let \(\mathcal{A}\) be some discrete time algorithm for the unfair two state problem. The adversary does not necessarily know which state \(\mathcal{A}\) is in, but he knows \(\mathcal{A}\)'s distribution. Let \(p_\mathcal{A}\) be the probability that the algorithm is in \(u\). The adversary produces a sequence of tasks and simulates \textsc{Two Stable} on them. Define \(\hat{\nu}\) and \(p(x)\) as in the previous section. Let \(m\) be a positive integer and \(\varepsilon = 1/m\). Let \(\mu\) be a positive real number such that
\[
\mu \leq p(\varepsilon), \quad 1 - \mu \geq p(1 - \varepsilon). \tag{5}
\]
The adversary behaves as follows:

1. If $p_A > 1 - \mu$ then charge $e$ to $u$ and 0 to $v$, else
2. If $p_A < \mu$ then charge $e$ to $v$ and 0 to $u$, else
3. If $p_A \geq p(\hat{w})$ then charge $e$ to $u$ and 0 to $v$, else
4. Charge $e$ to $v$ and 0 to $u$.

Rules 1 and 2 are necessary to ensure that the algorithm incurs unbounded cost. Consider the following algorithm: Stay in $u$ with probability 1 until $p(\hat{w}) = 0$. Then halve the probability of being in $u$ at each task. Without Rule 2, the adversary charges $u$ ad infinitum. The cost incurred by the algorithm is constant, since the probabilities form a convergent geometric series.

We compare the cost incurred by $\mathcal{A}$ to that incurred by Two Stable. We make the assumption that at the end of the request sequence, the algorithm matches its distribution to that of Two Stable. The algorithm can accomplish this with an extra cost of at most $\gamma$, which is constant with respect to the length of the request sequence. This assumption simplifies the analysis.

We break the request sequence into phases. A phase is a maximal sub-sequence of consecutive tasks where exactly one of $u$ or $v$ is charged. If state $x \in \{u, v\}$ is charged we call the phase an $x$ phase. The entire request sequence may be divided into $u$ and $v$ phases. We further classify phases. In a dominated phase, the state being charged is dominated before the last task of the phase. Phases which are not dominated phases are undominated phases.

Consider first an arbitrary undominated $u$ phase. Let $\ell$ be the number of tasks in the phase. Let $y_0 = y$ and $y_i = p_A$ after the $i$th task of the phase arrives. Note that by definition of the adversary we have

$$y_0 \geq p(x_0) - \mu \quad y_1 \geq p(x_1) \quad \text{for} \quad 0 < i < \ell \quad y_\ell \leq p(x_\ell) + \mu.$$

This situation is illustrated in Fig. 1.

The moving cost incurred by $\mathcal{A}$ in processing task $i$ is $\gamma |y_i - y_{i-1}|$. Let $a = p(x_i) + \mu - y_\ell$. The total moving cost incurred by $\mathcal{A}$ during the phase is at least $\gamma (y_0 - y_\ell) \geq \gamma (p(x_0) - p(x_\ell) - 2\mu + a)$; i.e., the moving cost incurred by $\mathcal{A}$ is at least as great as that incurred by Two Stable, minus $\gamma 2\mu$. For each of the tasks $1, ..., \ell - 1$, the stationary cost incurred by $\mathcal{A}$ is at least

$$y_i x_i \geq p(x_i) x_i \geq \int_{x_{i-1}}^{x_i} p(z) dz - exP,$$

where

$$P = \max_{-1 \leq x \leq 1} p(x) - p(x + e). \quad (6)$$
Note that $p$ is monotone decreasing, so $p(x) - p(x + \epsilon)$ is positive. The stationary cost for the last task is at least

$$y_l := \int_{x_{l-1}}^{x_l} p(z) \, dz =: P + a.$$ 

Let $X$ be the total cost (moving and stationary) incurred by Two Stable during the phase. Putting together the above facts, we conclude that the total cost to $A$ is at least

$$X \geq \int_{x_0}^{x_l} (p(x) - a) \, dx \geq \int_{x_{l-1}}^{x_l} p(z) \, dz - \epsilon x (P + a).$$

As long as $\epsilon \leq \gamma / \alpha$ this is at least $X - \epsilon x P - 2\mu \gamma$.

Now consider a dominated $u$ phase. Let $k$ be the least index $i$ for which $x_i = 1$. Since the phase is dominated such a $k < \ell$ must exist. Since only $u$ is charged during the phase we know, by definition of the work function, that if $x_i = 1$ then we have $x_j = 1$ for $k \leq i \leq \ell$. In other words, tasks $1, \ldots, k$ cause $\hat{w}$ to change, while tasks $k + 1, \ldots, \ell$ do not. The total moving cost incurred by $A$ during the phase is at least $\gamma (y_0 - y_k) \geq \gamma (p(x_0) - p(x_k) - 2\mu)$. For each of the tasks $1, \ldots, k$, the stationary cost incurred by $A$ is at least

$$y_i := \int_{x_{i-1}}^{x_i} p(z) \, dz - \epsilon x P.$$
The cost incurred by \( A \) for tasks \( k+1, \ldots, \ell \) is at least \( (\ell-k-1)\mu \alpha \). Let \( X \) be the total cost (moving and stationary) incurred by \textsc{Two Stable} for tasks 1, \ldots, \( k \). The total cost to the algorithm is at least

\[
X - k\alpha \beta P - 2\gamma \mu \geq \ell - (k-1)\mu \alpha.
\]

The analysis of \( v \) phases is almost identical. The cost to \( A \) for an undominated phase is at least \( X - \ell \alpha \beta P - 2\gamma \mu \) as long as \( \varepsilon \leq \gamma \beta \), while the cost for a dominated phase is at least

\[
X - k\alpha \beta P - 2\gamma \mu + (\ell - k - 1)\mu \beta \varepsilon.
\]

Let \( X_i \) be the total cost (moving and stationary) incurred by \textsc{Two Stable} for tasks 1, \ldots, \( k_i \). The total cost to the algorithm is at least

\[
X - k\alpha \beta P - 2\gamma \mu + (\ell - k - 1)\mu \beta \varepsilon.
\]

Recall from the previous section, that we derived the distribution \( p \) so that the amortized cost to \textsc{Two Stable} is exactly \( \varepsilon/2 \) whenever \( \hat{w} \) changes by \( \varepsilon \). Let \( \phi_i \) be the value of the potential function at the end of the \( i \)th phase. Let \( L = \sum_{i=1}^{N} (\ell_i - k_i - 1) \) By the analysis of the previous section we have

\[
\sum_{i=1}^{N} (X_i - k_i \alpha \beta P + Y_i) - \gamma = \sum_{i=1}^{N} (X_i + Y_i) - n\alpha \beta P - \gamma.
\]

Thus, we have

\[
\sum_{i=1}^{N} (\varepsilon \alpha \beta P + Y_i) - \gamma = \sum_{i=1}^{N} (X_i + Y_i) - n\alpha \beta P - \gamma.
\]

Recall from the previous section, that we derived the distribution \( p \) so that the amortized cost to \textsc{Two Stable} is exactly \( \varepsilon/2 \) whenever \( \hat{w} \) changes by \( \varepsilon \). Let \( \phi_i \) be the value of the potential function at the end of the \( i \)th phase. Let \( L = \sum_{i=1}^{N} (\ell_i - k_i - 1) \) By the analysis of the previous section we have

\[
\sum_{i=1}^{N} (\varepsilon \alpha \beta P + \phi_i - \phi_{i-1} + Y_i) - n\alpha \beta P - \gamma
\]

\[
= \varepsilon - 2\gamma \mu \frac{n \varepsilon}{2} + \phi_N - \phi_0 - n\alpha \beta P + L\mu \beta \varepsilon - \gamma
\]

\[
\geq \varepsilon - 2\gamma \mu P - 2\gamma \mu \frac{n \varepsilon}{2} - 2\phi_{\text{max}} + L\mu \beta \varepsilon - \gamma
\]

\[
\geq \varepsilon(1 - 2\mu P - 2\mu) \min \{ w(u), w(v) \} + L\mu \beta \varepsilon - \frac{7\varepsilon}{2} - \gamma.
\]

**Lemma 2.3.** When \( \varepsilon = \beta \) we have \( P = \varepsilon/2 \), and when \( \varepsilon > \beta \),

\[
P \leq \frac{\varepsilon \beta}{2}.
\]
Proof. Note that since $p$ is a continuous monotone decreasing function of $x$ for $-1 \leq x \leq 1$ we have

$$\max_{-1 \leq x \leq 1} \{ p(x) - p(x + \epsilon) \} \leq \epsilon \cdot \max_{-1 \leq x \leq 1} -\frac{dp(x)}{dx}.$$

Therefore, it suffices to show that $\max_{x} -\frac{dp(x)}{dx} \leq c/2$. For $\alpha > \beta$, we find that

$$-\frac{dp(x)}{dx} = \frac{(\alpha - \beta)e^{(x - \beta)(x - 1)/(2\gamma)}}{e^{\alpha/\gamma} - e^{\beta/\gamma}},$$

which is maximized at

$$\frac{(\alpha - \beta)e^{\alpha/\gamma}}{2\gamma(e^{\alpha/\gamma} - e^{\beta/\gamma})},$$

when $x = 1$. Clearly this is at most $c/(2\gamma) \leq c/2$.

In summary, the competitive ratio of $\mathcal{A}$ is at least $c(1 - c\alpha - 2\mu)$ when $\alpha > \beta$ and at least $c(1 - \epsilon - 2\mu)$ when $\alpha = \beta$. Since $\epsilon$ and $\mu$ are arbitrarily small, the competitive ratio of any algorithm is at least $c$.

Suppose we wish to use our adversarial strategy to get a sequence of tasks with optimal offline cost $x$, and cost to $\mathcal{A}$ of $c' \cdot x - \Delta$, where $c' < c$ and $\Delta = 7/2c + \gamma$. We accomplish this as follows:

1. Pick $\alpha > 0$ and $\mu > 0$ so that $c(1 - c\alpha - 2\mu) \geq c'$. The values of $\epsilon$ and $\mu$ must also satisfy $\epsilon \leq \alpha/\gamma$ and (5).
2. While $\mathcal{A}$’s cost is less than $c' \cdot x - \Delta$:
   (a) Generate a task using the adversarial strategy.
3. Generate a sequence of tasks which increases the optimal offline cost to $x$.

After Step 2 the optimal offline cost is at most $x$, and so the after Step 3 the optimal offline cost is $x$.

3. THE DECOMPOSITION THEOREM

We define the diameter of a nonempty set of states $X$ to be

$$D(X) = \max_{x, y \in X} d(x, y).$$

We define the distance between a state $x$ and a set of states $Y$ to be

$$d(x, Y) = \min_{y \in Y} d(x, y).$$
The distance between two sets of states $X$ and $Y$ is

$$d(X, Y) = \min_{x \in X, y \in Y} d(x, y).$$

We consider task systems for which the the set of states $S$ can be partitioned into sets $U$ and $V$ such that

$$\theta = \frac{d(U, V)}{\max \{D(U), D(V)\}}.$$ 

is greater than 1. I.e. $U$ and $V$ are widely separated and small relative to the overall diameter of $S$. We call such a space $\theta$-decomposable. Without loss of generality, we assume that $d(U, V) = 1$. It is not hard to show, using the triangle inequality, that

$$D(S) \leq 1 + 2\theta.$$ 

Let $X$ be any nonempty subset of $S$. We extend the definition of the work function as

$$w_0(X, x) = d(x, s_1)$$

$$w_{i+1}(X, x) = \min_{y \in X} \{ w_i(y) + T_{i+1}(y) + d(x, y) \},$$

for any $x \in X$. Note that $\min_{x \in X} w_i(X, x)$ is the optimal offline cost to process the first $i$ tasks, always staying in $X$. Also note that $w(x) = w(S, x)$.

For $X$ a nonempty subset of $S$, we define

$$\tau_i(X) = \min_{x \in X} w_i(X, x) - \min_{x \in X} w_{i-1}(X, x)$$

We wish to bound the optimal offline cost using $w(U, u)$ and $w(V, v)$. Let $1 \leq t_1 < t_2 < \ldots < t_k \leq \ell$ be integer transition times. We consider the minimum cost incurred by an algorithm which makes transition from $U$ between $V$ at times $t_1, \ldots, t_k$ and possibly at time 0. Let $X \in \{ U, V \}$. Let $\text{cost}_{\text{opt}}(\sigma, X, t_1, \ldots, t_k)$ be the minimum cost incurred by an algorithm for task sequence $\sigma$ which:

1. Is allowed to make a transition at time 0.
2. Processes tasks $1, \ldots, t_1$ in $X$.
3. For any positive $t$ the algorithm moves between $U$ and $V$ if and only if $t = t_i$ for some $i \in \{1, \ldots, k\}$.

Let $f(X, i, j)$ be the minimum cost incurred by any algorithm which stays in $X$ while processing tasks $T_{t_1}, \ldots, T_j$. The algorithm is allowed to be in any state in $X$ before $T_i$, and to be in any state in $X$ after $T_j$. Let $X \in \{ U, V \}$ and let $Y$ be the other member of $\{ U, V \}$. Define
\[ B(\sigma, X, t_1, \ldots, t_k) = \min_{x \in X} w_t'(X, x) + 1 + f(Y, t_1 + 1, t_2) + 1 + f(X, t_2 + 1, t_3) + \cdots \\
A(\sigma, X, t_1, \ldots, t_k) = \min_{x \in X} w_t'(X, x) + 1 + \frac{2}{\theta} + f(Y, t_1 + 1, t_2) + 1 + \frac{2}{\theta} + f(X, t_2 + 1, t_3) + \cdots. \]

We have

\[ B(\sigma, X, t_1, \ldots, t_k) \leq \text{cost}_{\text{opt}}(\sigma, X, t_1, \ldots, t_k) \leq A(\sigma, X, t_1, \ldots, t_k). \]

We bound \( f(X, i, j) \). Define

\[ g(X, i, j) = \sum_{k=i}^{j} \tau_k(X) = \min_{x \in X} w_j(X, x) - \min_{x \in X} w_{i-1}(X, x). \]

\( g(X, i, j) \) is the minimum cost incurred by any algorithm which stays in \( X \) while processing tasks \( T_i, \ldots, T_j \), given that the algorithm is allowed to end processing in any state, but is charged

\[ w_{i-1}(X, x) - \min_{x \in X} w_{i-1}(X, x) \leq \frac{1}{\theta}, \]

to start in state \( x \). So clearly

\[ g(X, i, j) - \frac{1}{\theta} \leq f(X, i, j) \leq g(X, i, j). \]

We also note that

\[ \min_{x \in X} w_{t_i}(X, x) = g(X, 1, t_1) + \min_{x \in X} w_0(X, x) = g(X, 1, t_1) + \min_{x \in X} d(x, s_1) = g(X, 1, t_1) + d(X, s_1). \]

Therefore, we have

\[ B(\sigma, X, t_1, \ldots, t_k) = d(X, s_1) + g(X, 1, t_1) + 1 - \frac{1}{\theta} + g(Y, t_1 + 1, t_2) + \cdots \\
A(\sigma, X, t_1, \ldots, t_k) = d(X, s_1) + g(X, 1, t_1) + 1 + \frac{2}{\theta} + g(Y, t_1 + 1, t_2) + \cdots. \]
Clearly
\[ \text{cost}_{\text{opt}}(\sigma) = \min_{X, k, t_1 \ldots, t_k} \text{cost}_{\text{opt}}(\sigma, X, t_1, \ldots, t_k), \]
and therefore
\[ \min_{X, k, t_1 \ldots, t_k} B(\sigma, X, k, t_1, \ldots, t_k) \leq \text{cost}_{\text{opt}}(\sigma) \leq \min_{X, k, t_1 \ldots, t_k} A(\sigma, X, t_1, \ldots, t_k). \]

It is easily seen that these upper and lower bounds can be computed using the dynamic programming recurrences
\[
\begin{align*}
\omega_0(X) &= d(s_1, X) \\
\omega_{i+1}(X) &= \min_{Y \subseteq \{U, V\}} \left\{ \omega_i(Y) + \tau_{i+1}(Y) + \left(1 - \frac{1}{\theta}\right)d(X, Y) \right\} \\
\end{align*}
\]
and
\[
\begin{align*}
\tilde{\omega}_0(X) &= d(s_1, X) \\
\tilde{\omega}_{i+1}(X) &= \min_{Y \subseteq \{U, V\}} \left\{ \tilde{\omega}_i(Y) + \tau_{i+1}(Y) + \left(1 + \frac{2}{\theta}\right)d(X, Y) \right\}. \\
\end{align*}
\]

In fact,
\[
\begin{align*}
\min_{X, k, t_1 \ldots, t_k} B(\sigma, X, t_1, \ldots, t_k) &= \min\{ \omega_r(U), \omega_r(V) \} \\
\min_{X, k, t_1 \ldots, t_k} A(\sigma, X, t_1, \ldots, t_k) &= \min\{ \tilde{\omega}_r(U), \tilde{\omega}_r(V) \}. \\
\end{align*}
\]

The functions \( \omega \) and \( \tilde{\omega} \) have many of the properties of work functions. Note that
\[
|\omega(U) - \omega(V)| \leq 1 - \frac{1}{\theta}, \quad |\tilde{\omega}(U) - \tilde{\omega}(V)| \leq 1 - \frac{2}{\theta}.
\]

We extend the definition of domination to \( \omega \) and \( \tilde{\omega} \) in the obvious way. For all \( i \), we have
\[
\omega_i(X) - \omega_{i-1}(X) \leq \tau_i(X).
\]

Further, if \( \omega_i(X) - \omega_{i-1}(X) \leq \tau_i(X) \) then \( \omega_i(X) \) is dominated. These same facts are true of \( \tilde{\omega} \).

Suppose that we have a randomized algorithm \( \mathcal{A} \) for the task system \((U, d)\) such that for any sequence of tasks \( \sigma \),
\[
E[\text{cost}_{\mathcal{A}}(\sigma)] \leq x \cdot \text{cost}(\sigma) + A
\]
for some constant $A$. Further suppose that for $(V, d)$ we have a randomized algorithm $B$ such that for any sequence of tasks $\sigma$,

$$E[\text{cost}_{\sigma}(\sigma)] \leq \beta \cdot \text{cost}(\sigma) + A.$$  

The first part of the decomposition theorem is as follows:

**Theorem 3.1.** Let $\gamma = (\theta + 2)/(\theta - 1)$. For $\theta > 1$, for any positive integer $m$ there exists an algorithm for the space $(S, d)$ which is

$$\frac{x e^{\alpha / \gamma} - \beta e^{\beta / \gamma}}{e^{\alpha / \gamma} - e^{\beta / \gamma}} + \frac{x^2}{m} + 4mA$$

competitive when $\alpha > \beta$, and at most $\alpha + \gamma + \alpha/m + 4mA$ when $\alpha = \beta$.

**Proof.** We define

$$\hat{\psi} = \frac{\psi(U) - \psi(V)}{1 - 1/\theta}.$$  

Note that $-1 \leq \hat{\psi} \leq 1$. Effectively, we charge the adversary $1 - 1/\theta$ for moving between $U$ and $V$. The our algorithm is charged $1 + 2/\theta \geq D(S)$. Therefore we set $\gamma = (\theta + 2)/(\theta - 1)$.

We describe an algorithm for $(S, d)$ which we call the Discrete Two Stable algorithm. Let $m$ be a positive integer. Define $\varepsilon = 1/m$. The algorithm behaves as follows: When $\hat{\psi} = \varepsilon$ for $i = -m, -m + 1, \ldots, m$ the algorithm changes its distribution so that the probability that it is in $U$ is $p(i\varepsilon)$. At other times, the algorithm’s distribution remains fixed. Within $U$ and $V$ the algorithm runs $A$ and $B$, respectively. $A$ and $B$ are oblivious of the fact that they are running in a subspace of a larger space. Let $g$ be the probability that Discrete Two Stable is in $U$. Let $f(u)$ be the probability that Discrete Two Stable is in state $u$. Then Discrete Two Stable is in $u$ with probability $q \cdot f(u)$. Similarly, Discrete Two Stable is in $v$ with probability $(1 - q) \cdot g(u)$, where $g(u)$ is the probability that $B$ is in state $v$.

We show that Discrete Two Stable algorithm is

$$c(1 + 2P) + 4A$$

competitive for $(S, d)$, where $c$ is defined by (4) and $P$ is defined by (6). Given this, the theorem will follow from Lemma 2.3.

Define an event to be a time at which the algorithm’s distribution changes. We analyze each event. Let $y$ be the amount of useful work incurred between the current event and the previous event. Let $y_1$ be the amount by which $\psi(U)$ increases and $y_2$ be the amount by which $\psi(V)$ increases so that $y = y_1 + y_2$. Note that $y_1 \geq \varepsilon$. The adversary’s cost for the event is $y/2$. Consider first an event where $\hat{\psi}$ achieves a value of $x' = i\varepsilon$ and where at the previous event $\hat{\psi}$ achieved a value of $x = (i - 1)\varepsilon$. We bound the competitive ratio for each event. The cost incurred by the algorithm is at most

$$x y_1 p(x) + A + \beta y_2 (1 - p(x)) + A + \gamma(p(x) - p(x')).$$
Since the adversary’s cost is \( y_2 \), in order to maximize the competitive ratio, the adversary sets \( y_1 = \varepsilon \) and \( y_2 = 0 \). (Assuming that \( c \geq \varepsilon \).) The algorithm’s cost is therefore

\[
\alpha(x' - x) p(x) + 2A + \gamma(p(x) - p(x'))
\]

\[
\leq \alpha \int_x^{x'} p(x) + \alpha|p(x) - p(x')| \varepsilon + 2A + \gamma(p(x) - p(x'))
\]

\[
\leq \alpha \int_x^{x'} p(x) + \gamma(p(x) - p(x')) + 2A + \alpha Pe.
\]

If one ignores the \( 2A + \alpha Pe \) term, this is the same as (1), and the same amortized analysis of the cost, using the same potential function, follows. The amortized cost incurred by the algorithm is at most

\[
c = \frac{\varepsilon}{2} + \alpha Pe + 2A.
\]

Since the adversary’s cost is \( \varepsilon/2 \), the competitive ratio is \( c = 2\alpha P + 4mA \). For the event \( \hat{w} \) achieves \( (i - 1)\varepsilon \) and the value at the previous event was \( i\varepsilon \), the cost incurred by the algorithm is upper bounded by

\[
\beta \int_x^{x'} (1 - p(x)) + \gamma(p(x') - p(x)) + 2A + \beta Pe,
\]

which, omitting the last term, is the same as (2). Once again, the same amortized analysis is used and the competitive ratio is at most \( c + 2\beta P + 4mA \leq c + 2\alpha P + 4mA \). So in all cases, the competitive ratio is at most \( c + 2\alpha P \leq c(1 + 2P + 4mA) \). In fact, the cost for any sequence \( \sigma \) is at most

\[
(c(1 + 2P) + 4mA) \min\{w(U), w(V)\} + \frac{7c}{2}
\]

\[
\leq (c(1 + 2P) + 4mA) \text{cost}_{\text{opt}}(\sigma) + O(c \cdot D(S)).
\]

Suppose that we have an adversarial strategy \( \mathcal{A} \) for the task system \((U, d)\) such that for any algorithm \( \mathcal{C} \)

\[
E[\text{cost}_x(\sigma)] \geq \alpha \cdot \text{cost}(\sigma) - \Delta,
\]

where \( \sigma \) is the sequence of tasks produced by \( \mathcal{A} \) and \( \Delta \) is some constant. Further suppose that for \((V, d)\) we have an adversarial strategy \( \mathcal{B} \) such that for any algorithm \( \mathcal{C} \):

\[
E[\text{cost}_x(\sigma)] \geq \beta \cdot \text{cost}(\sigma) - \Delta.
\]

The second part of the decomposition theorem is:
Theorem 3.2. Let $\mathcal{C}$ be an algorithm for $(S, d)$. Let $\gamma = \theta/(\theta + 2)$. For $\theta > 2$, for any positive integer $m$, the competitive ratio of $\mathcal{C}$ is at least
\[
\frac{\alpha e^{\alpha/\gamma} - \beta e^{\beta/\gamma}}{e^{\alpha/\gamma} - e^{\beta/\gamma}} - \frac{\alpha^2}{m} - 2mA
\]
when $\alpha > \beta$, and at least $\alpha + \gamma - \alpha/m - 4mA$ when $\alpha = \beta$.

Proof. We define
\[
\hat{w} = \frac{w(U) - w(V)}{1 + 2/\theta}.
\]
Note that $-1 \leq \hat{w} \leq 1$. $\mathcal{C}$ is charged 1 for moving between $U$ and $V$. So we set $\gamma = 1/(1 + 2/\theta) = 0/(\theta + 2)$.

Let $\varepsilon = 1/m$. Let $\mu$ be a positive real satisfying (5). Let $p_\varepsilon$ be the probability that $C$ is in $U$. Our adversary behaves as follows. If $p_\varepsilon \geq 1 - \mu$ or if $p_\varepsilon \geq p(\hat{w})$ and $p_\varepsilon \geq \mu$ then apply $A$ within $U$ until $\min_{u \in U} \hat{w}(U, u)$ increases by $\varepsilon$. Otherwise, apply $B$ within $V$ until $\min_{v \in V} \hat{w}(V, v)$ increases by $\varepsilon$.

The analysis differs very little from that of Section 2.3. We require $\mathcal{C}$ to be in $U$ with probability $p(\hat{w})$ at the end of the task sequence. The algorithm incurs an extra cost of at most $1 + 2/\theta$. We break the task sequence into phases, as in the analysis of Section 2.3. Each phase, instead of consisting of individual tasks, consists of applications of $A$ or $B$. We must take $A$ into account in the cost of a phase. For an undominated phase we get a total cost of
\[
X - \ell \alpha P - \ell A - 2\gamma \mu
\]
as long as $\varepsilon \leq \gamma/\alpha$. For a dominated phase we get a cost of
\[
X - \ell \alpha P + (\ell - k - 1) \mu \beta - \ell A - 2\gamma \mu.
\]
We sum this over all phases, as in Section 2.3 and get
\[
(c - 2\alpha P - 2\mu \gamma) \frac{nc}{2} - 2c + L\mu \beta - 1 - \frac{2}{\theta} - nA
\]
\[
= (c - 2\alpha P - 2\mu \gamma - 2mA) \frac{nc}{2} - 2c + L\mu \beta - 1 - \frac{2}{\theta}
\]
\[
\geq (c - 2\alpha P - 2\mu \gamma - 2mA) \min\{\hat{w}(U), \hat{w}(V)\} + L\mu \beta - \frac{5c}{2} - 1 - \frac{2}{\theta}
\]
\[
\geq (c - 2\alpha P - 2\mu \gamma - 2mA) \text{cost}_{\text{opt}}(\sigma) + L\mu \beta - O(\epsilon \cdot D(S)).
\]
The theorem follows from Lemma 2.3 and the fact that $\mu$ is arbitrarily small. •
4. APPLICATIONS OF THE DECOMPOSITION THEOREM

In (Blum et al. 1992) several applications of the decomposition theorem are stated, the most important being a lower bound for all spaces. We mention two applications of the decomposition theorem which are implied by (Blum et al. 1992) but not stated there.

The isosceles triangle has \( d(s_1, s_2) = 1 \) and \( d(s_1, s_3) = d(s_2, s_3) = x \). For the task system on this metric space we get an algorithm whose competitive ratio approaches

\[
1 + \frac{e}{e-1}
\]

as \( x \to \infty \). This is 1 more than the bound for 2-servers on the isosceles triangle (Karlin et al. 1994).

The balanced metric space \( B(n, \theta) \) for \( n \) a power of 2 is defined as follows:

1. \( B(1, \theta) \) consists of a single point.
2. \( B(2^i, \theta) \) consists of two copies of \( B(2^{i-1}, \theta) \), call them \( T \) and \( U \), such that

\[
d(t, u) = \theta^i,
\]

for all \( t \in T \) and \( u \in U \). When \( \theta = \Omega(\log^2 n) \), we set \( m = \lceil \sqrt{\theta} \rceil \) and apply the decomposition theorem recursively to show that the competitive ratio of \( B(n, \theta) \) is \( \Theta(\log n) \). As \( \theta \to \infty \) the competitive ratio approaches \( \log \sqrt{n} + 1 \).

Received January 27, 1997; final manuscript received May 31, 1998.

REFERENCES


Blum, A., Furst, M., and Tomkins, A., What to do with your free time: On-line algorithms for infrequent request, unpublished manuscript.


Chrobak, M., and Noga, J., unpublished manuscript.


