On Algorithm Design for Metrical Task Systems

William R. Burley[*] Sandy Irani[†]

January 20, 1996

Abstract

We extend the definition of *Metrical Task Systems*, introduced by Borodin, Linial and Saks in [4]. In the extended definition, a system is described by the underlying metric space of states $\mathcal{M}$ as well as a set of allowable tasks $\mathcal{T}$. Any request to an algorithm must be a member of $\mathcal{T}$. The extension makes the model powerful enough to completely characterize many important on-line problems. We consider methods of designing competitive algorithms given the description of a system $\langle \mathcal{M}, \mathcal{T} \rangle$. In particular, we show that it is PSPACE-hard to determine the behavior of a $c(\mathcal{M}, \mathcal{T})$-competitive algorithm, where $c(\mathcal{M}, \mathcal{T})$ is the best possible competitive ratio on $\langle \mathcal{M}, \mathcal{T} \rangle$. In addition, we show a simple, polynomial time algorithm for task systems $\langle \mathcal{U}_n, \mathcal{T} \rangle$ (where $\mathcal{U}_n$ is the uniform metric space on n nodes) that achieves a competitive ratio of $O(\log n \cdot c(\mathcal{M}, \mathcal{T}))$.

Keywords: Analysis of algorithms, on-line algorithms, competitive analysis, approximation algorithms, PSPACE-completeness, metrical task systems.

[*]Dept. of Computer Science and Engineering, University of California, San Diego. Research supported in part by NSF grants CCR-8911388 and CCR-92-15293. Current address: AT&T Bell Laboratories, Middletown, NJ 07748. Email: `william.burley@att.com`

[†]Supported in part by NSF Grant CCR-9309456 and GER-94-50142. Address: Department of Information and Computer Science, University of California, Irvine, CA 92717. Phone: (714) 824-6346. Email: `irani@ics.uci.edu`.

1 Introduction

Computer systems are often faced with the problem of performing a series of tasks where there is more than one way to perform a given task. The way a task is executed affects the cost of execution as well as the state of the system upon completion of the task. The state of the system in turn affects the cost of processing future tasks. Usually it is the case that these decisions must be made without any knowledge of the future demands that will be made on the system. An algorithm that bases its decisions only on past requests is said to be *on-line*. Many fundamental problems which arise in computer science share this basic paradigm. However, a unified approach to these problems has been elusive. Typically, techniques are ad-hoc and algorithm designers rely on special structure to solve the particular problem at hand. What is needed is a more systematic approach which can be used in the development of on-line algorithms for large classes of problems. To this end, we consider a general model of on-line systems and look at methods for designing algorithms given the description of a problem in this model. Our model is an extended version of the *Metrical Task Systems* model introduced by Borodin, Linial and Saks in [4]. This extended model is simple yet powerful enough to completely formulate many interesting on-line problems.

A *task system* is comprised of a finite set of states and distances between each pair of states. (We assume that the distances obey the triangle inequality and that the distance from a state to itself is 0). The system is said to be a *metrical task system* if the distances are symmetric. Thus, a metrical task system is defined by the underlying metric space $\mathcal{M}$. The set of states is denoted by $S^{\mathcal{M}}$ and the distance between any two states i and j in $S^{\mathcal{M}}$ is denoted $d^{\mathcal{M}}_{i,j}$. (We omit the superscript $\mathcal{M}$ when the metric space is clear from context). A sequence of *tasks* must be performed. A task is a non-negative vector indicating the cost of processing that task in each of the states; $T(i)$ is the cost of processing the task T in state i. The input is a sequence of tasks, $\sigma = T^1, T^2, \ldots, T^N$. The objective of the algorithm is to determine a state in which to process each task, balancing the cost of moving with the cost of processing the tasks. The algorithm produces a schedule π where $\pi(j)$ is the state in which T^j is processed. If the algorithm is on-line, then $\pi(j)$ is a function only of $T^1, \ldots, T^j$. The cost of a schedule π on σ is the sum of the cost of moving from state to state (the *moving costs*) and the cost of processing the tasks (the *stationary costs*):

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

$\pi(0)$ is some fixed initial state. We denote by $A(\sigma)$ the schedule produced by algorithm A on input σ. The *cost of algorithm A on σ* denoted $cost_A(\sigma)$ is $cost(A(\sigma), \sigma)$. The cost of the optimal off-line algorithm for the sequence π is

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

We evaluate an on-line algorithm by comparing its performance to that of the optimal off-line algorithm. An on-line algorithm A is said to be c-competitive on $\mathcal{M}$ if there is a constant b such that for all σ,

$$cost_A(\sigma) \leq c \cdot cost(\sigma) + b.$$

The *competitive ratio* of A on $\mathcal{M}$ denoted $c_A(\mathcal{M})$ is the infimum over all c such that A is c-competitive on $\mathcal{M}$. For a given task system $\mathcal{M}$, the best possible competitive ratio achievable on that system is denoted $c(\mathcal{M})$. An on-line algorithm for $\mathcal{M}$ which is $c(\mathcal{M})$-competitive is said to *strongly competitive*. This means of evaluating on-line algorithms called *competitive analysis* was initiated by Sleator and Tarjan in [12] and has become the standard measuring stick used by the vast majority of recent work in on-line algorithms.

The results by Borodin *et al.* in [4] on deterministic algorithms for metrical task systems appear to be conclusive: they show that for any metrical task system $\mathcal{M}$, $c(\mathcal{M}) = 2n - 1$ where n is the number of states in $S^{\mathcal{M}}$. However, these results also appear to be in disagreement with results for particular on-line problems which fall in the framework of metrical task systems. Let's take paging as a specific example:

A paging algorithm manages a two-level store consisting of a fast memory that can hold k pages, and a large slow memory. The algorithm is presented with a sequence of requests to virtual memory pages. If the page requested is in fast memory it incurs no cost; but if not, the algorithm must bring it in to fast memory at unit cost, and decide which of the k pages currently in fast memory to evict in order to make room for it. When paging is expressed as a task system, the set of states is the set of all possible contents of the fast memory, (i.e., all k-sets of virtual memory pages). The distance between two states is the number of pages that are in one k-set and not in the other k-set. A request for a page x corresponds to the task vector whose entry is 0 for every k-set that contains x and ∞ for every k-set that does not contain x. Thus, when paging is phrased as a task system, the number of states is $n = \binom{p}{k}$ where p is the number of virtual memory pages. Since the number of pages p is generally very large, the competitive ratio of $2\binom{p}{k} - 1$ implied by the Borodin *et al.* result ([4]) compares miserably with the known k-competitive algorithm for the paging problem in [12].

The discrepancy lies in the fact that every request in the paging problem corresponds to a task in the corresponding task system, but not every non-negative vector represents a request in the paging problem. The lower bound of $2n - 1$ for task systems assumes an adversary which can request any non-negative vector. For this reason, task systems have been criticized since the model is not sensitive enough to describe many interesting on-line problems. We address this weakness by including in the description of the system the underlying space of configurations $\mathcal{M}$ as well as a set of *allowable tasks* $\mathcal{T}$ where $\mathcal{T}$ is a subset of all non-negative n-vectors. With this feature added to the model, metrical task systems are now powerful enough to completely formulate a large class of the on-line problems of interest. Examples include: the k-server problem, paging, layered graph traversal, operations on dynamic data structures, and maintaining data in a multiprocessing environment ([1, 2, 7, 8, 10, 12, 13, 15]).

When $\mathcal{T}$ has a special structure or limited range, a competitive ratio better than $2n - 1$ can often be achieved. An on-line algorithm A is said to be c-competitive on $\langle \mathcal{M}, \mathcal{T} \rangle$ if there is a constant b such that for all $\sigma \in \mathcal{T}^*$,

$$cost_A(\sigma) \leq c \cdot cost(\sigma) + b.$$

The *competitive ratio* of A on $\langle \mathcal{M}, \mathcal{T} \rangle$ denoted $c_A(\mathcal{M}, \mathcal{T})$ is the infimum over all c such that A is c-competitive on $\langle \mathcal{M}, \mathcal{T} \rangle$. For a given task system $\mathcal{M}$, the best possible competitive ratio achievable on that system is denoted $c(\mathcal{M}, \mathcal{T})$. An algorithm which is $c(\mathcal{M}, \mathcal{T})$-competitive is said to be *strongly competitive*.

With this framework in place, the most natural questions are: given a task system $\langle \mathcal{M}, \mathcal{T} \rangle$, what is $c(\mathcal{M}, \mathcal{T})$? what is the algorithm that achieves this competitive ratio? We would like an algorithm that can compute $c(\mathcal{M}, \mathcal{T})$, given $\langle \mathcal{M}, \mathcal{T} \rangle$. Or, better yet, we would like a general method for designing a strongly competitive algorithm for $\langle \mathcal{M}, \mathcal{T} \rangle$ based on the description of the system. A 'universal' algorithm of this kind would represent the first step in establishing a systematic and unified approach to solving new on-line problems.

In Section 3 we show that developing a universal algorithm which is always optimal among on-line algorithms is likely to be computationally infeasible even when the task systems are limited. We call a task which has only entries that are 0 or ∞, a 0-∞ task. A task system whose set of allowable tasks $\mathcal{T}$ consists only of 0-∞ tasks is called a 0-∞ *task system*. The metric space on n states where all distances are 1 is called the *uniform metric space* and is denoted $\mathcal{U}_n$. A task system whose metric space is uniform is called a *uniform task system*. We prove the following result:

Theorem 1 *Given a 0-∞, uniform task system $\langle \mathcal{U}_n, \mathcal{T} \rangle$, it is PSPACE-complete to determine $c(\mathcal{U}_n, \mathcal{T})$.*

This theorem shows that the problem of computing $c(\mathcal{U}_n, \mathcal{T})$ is PSPACE-complete, but does not directly address the issue of devising a universal algorithm for any task system. Although the term universal is not well defined, intuitively, we would like a single algorithm which uses the description $\langle \mathcal{M}, \mathcal{T} \rangle$ of the task system to determine its behavior and whose competitive ratio is close to $c(\mathcal{M}, \mathcal{T})$. We would also like the algorithm to be efficient. One difficulty with the problem as it is stated is that the constant b in the definition of competitiveness is allowed to depend on $\langle \mathcal{M}, \mathcal{T} \rangle$ (although not on σ) and could be exponential in n. In such a case, an algorithm could potentially determine a strongly competitive algorithm in the first 2^n requests and behave according to this algorithm for the remainder of the sequence. For this reason, we would like to restrict the additive constant used in the definition of competitiveness to "reasonable" values. Letting d be the maximum distance between any two states in $\mathcal{M}$, it follows from results of Chrobak and Larmore [6] that a $c(\mathcal{M}, \mathcal{T})$-competitive algorithm with additive constant at most $2dn$ exists. (Although Chrobak and Larmore's results are for 0-∞ task systems, the essential property, which is the Lipshitzness of the minimal potential, does not depend on the task costs and also holds for non-0-∞ task systems.) Thus, we restrict the definition of competitiveness so that the additive constant must be polynomial in n and d. With this restricted definition, Theorem 1 can be extended to show that unless P = PSPACE, there is no universal algorithm which works in time polynomial in the description of the system and is guaranteed to be $c(\mathcal{M}, \mathcal{T})$-competitive for every $\langle \mathcal{M}, \mathcal{T} \rangle$.

In light of this result, we seek a universal algorithm which can approximate a strongly competitive algorithm for a given task system, and which works in time polynomial in the description of the system. In Section 2, we obtain results in this direction for the important special case of uniform task systems. We define a simple algorithm GREEDY and prove the following theorem:

Theorem 2 *Let $\langle \mathcal{U}_n, \mathcal{T} \rangle$ be a uniform task system on n states, then*

$$c_{GREEDY}(\mathcal{U}_n, \mathcal{T}) = O(\log n \cdot c(\mathcal{U}_n, \mathcal{T})).$$

The term 'universal' is borrowed from [3] where a single paging algorithm is devised that performs well under a variety of restrictions on the input sequence.

Some mention should be made at this point of how the task system is represented. We have been assuming that a task system is represented by an explicit description of the distances between states and the set of all allowable tasks. This representation certainly allows for expressing the most general class of systems. Some task systems, however, may have some regularity which allows for a more succinct representation. Denote the length of an explicit representation of the task system by $l(\mathcal{M}, \mathcal{T})$. The hardness result of Section 3 holds even if the complexity is defined as a function of $l(\mathcal{M}, \mathcal{T})$. The algorithm in Section 2 can be implemented in time polynomial in $l(\mathcal{U}_n, \mathcal{T})$. The implementation may possibly be improved in special cases when there is some symmetry in $\langle \mathcal{M}, \mathcal{T} \rangle$. Of course the representation, whether explicit or implicit, must be decodable in polynomial time so that the decisions the algorithm must make can be efficiently computable.

1.1 Related Work

In their original paper on metrical task systems, Borodin *et al.* ([4]) recognized the need to investigate how different restrictions on the set of allowable tasks affect the competitive ratio. The extended model of metrical task systems that we use was first described by Manasse, McGeoch, and Sleator in [9]. While a great deal of work has been done analyzing individual problems, less effort has been made in the development of general techniques for the design of competitive algorithms for any metrical task system.

Determining the optimal competitive ratio for a given on-line problem and constructing a strongly competitive algorithm for that problem are issues which have been studied for general task systems as well as for certain other general models of on-line problems. In [9], Manasse *et al.* give a decision procedure that, for a metrical task system $\langle \mathcal{M}, \mathcal{T} \rangle$ with rational distances and task costs and any rational c, determines if $c(\mathcal{M}, \mathcal{T}) \leq c$. Their decision procedure, which is exponential in the size of $\langle \mathcal{M}, \mathcal{T} \rangle$, also constructs a c-competitive algorithm if one exists. Results of a similar flavor are described by Chrobak and Larmore in [5] for a class of problems, more general than metrical task systems, called *on-line games*, and in [6] for the class of *metrical service systems*, which are essentially 0-∞ task systems. In [3], Borodin *et al.* have similar results for a paging problem where the input to the problem is restricted to be of a certain form. Our complexity results are of a different nature. We prove that in general these types of problems are difficult to solve, even for a fairly restricted class of task systems.

Given the difficulty of devising strongly competitive algorithms for general task systems, it is natural to ask whether universal algorithms exist that are efficient to implement, yet still achieve a competitive ratio close to optimal. We know of little previous work in this area. Universal algorithms for a paging problem with restricted input were considered in [3], however their problems are substantially different from those we consider here. We could also ask whether existing algorithms are reasonable candidates for the job. In particular, the work function algorithm (see [5]) is well-defined for any finite metrical task system, and is known to be optimal (i.e. $2n-1$-competitive) for the set of all possible tasks. Unfortunately, it is not hard to find examples, even for 0-∞ uniform task systems, where the competitive ratio of the work function algorithm is $\Omega(n)$ times worse than the optimal ratio. As an example, consider the 0-∞ task system $\langle \mathcal{U}_n, \mathcal{T} \rangle$ where $\mathcal{U}_n$ is the uniform metric space on the states $\{1, 2, \ldots, n\}$ and $\mathcal{T}$ is defined as follows. For this example we will represent a task by the set of states whose task cost is ∞. We let $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ where $T_i = \{i\}$ for $i = 1$ to $n-1$, and $T_n = \{2, 3, \ldots, n\}$. The optimal on-line algorithm moves only between the two states 1 and n and achieves a competitive ratio of 1. The work function algorithm, however, could potentially cycle through each state in turn resulting in a competitive ratio of only $n-1$.

2 A Universal Algorithm for Uniform Task Systems

In this section, we define a simple algorithm GREEDY for any uniform task system $\langle \mathcal{U}_n, \mathcal{T} \rangle$. GREEDY is an efficient algorithm, taking only polynomial time at each step (polynomial in $l(\mathcal{U}_n, \mathcal{T})$). We will show that GREEDY achieves a "good" competitive ratio, i.e., within a factor of $O(\log n)$ of the optimal on-line algorithm, for any $\langle \mathcal{U}_n, \mathcal{T} \rangle$.

Before defining GREEDY, we need to make an assumption about the tasks in $\mathcal{T}$. For any task $T \in \mathcal{T}$, we assume that all task costs are within a range of 2. More precisely, if $d = \min_s T(s)$, then $T(x) \leq d + 2$ for all x. We can make this assumption since the minimum cost to "serve" a task from a state cannot be more than the cost to move to the state which has cost d, serve the request, then move back.

We start with some intuition about GREEDY. GREEDY operates in phases. During each phase, GREEDY keeps track of a cost function v, where $v(s)$ is the "cost" of being in state s since the beginning of the phase. Set to 0 at the start of a phase, the cost function is updated for each task requested during the phase, however, the cost function is never increased above 1 for any state. A phase ends when the cost function reaches 1/2 for every state. This insures that any algorithm, even an off-line algorithm, will incur cost of at least 1/2 per phase. During a phase, GREEDY moves about the states of the metric space. The time between each move is called a *round*. During a round, GREEDY stays in a single state. The cost incurred by GREEDY during a round is simply the sum of the task costs for that state for the tasks requested during the round. A round ends when GREEDY incurs a cost of at least 1 since the beginning of the round. At the end of a round, GREEDY moves to a new state for the next round (we describe below how this state is chosen). For intuitive purposes, we can think of GREEDY as incurring a cost of approximately 2 for each round (1 to move plus the cost incurred during the round), although the actual cost may be more depending on the last request of each round. Thus, GREEDY's total cost for a phase is roughly proportional to the number of rounds in that phase. GREEDY can minimize its competitive ratio by minimizing the number of rounds in a phase (i.e., by minimizing the number of rounds it takes until the cost function is at least 1/2 at every state).

GREEDY attempts to minimize the number of rounds by making the cost function increase by as much as possible at each round. It does this as follows. At the end of every round GREEDY computes the "influence" of every state. The influence of a state is defined as the minimum total increase in the cost function (i.e., the increase in $\sum_s v(s)$) which would occur during the next round if GREEDY moved to that state for the round and the adversary then picked a sequence of tasks which caused the algorithm to move again. The influence is based on the sequence of tasks which results in the smallest possible increase in $\sum_s v(s)$ during the next round. The influence is only a lower bound on the increase in the cost function—the actual increase could be more depending on the tasks requested during the round. Note that the influence may differ for different states depending on the set of tasks $\mathcal{T}$, and it may change (decrease) after each round because of an increase in the cost function during the round. At the end of every round, GREEDY chooses to move to the state that has the largest influence for the next round. In other words, the algorithm estimates the worst state for the adversary based on looking ahead one round. It is in this sense that the algorithm is greedy. The strategy guarantees that the cost function will increase by at least the influence of the chosen state during the next round. Although this heuristic doesn't necessarily minimize the number of rounds, it can be shown that the number of rounds will be within an $O(\log n)$ factor of optimal.

We now define GREEDY more explicitly. At the beginning of a phase, GREEDY sets $v(s) = 0$ for all s. For each task T requested during a phase, the cost function is updated as follows. For v a cost function and T a task, we define $v \wedge T$, the *update of v by T*, by

$$(v \wedge T)(s) = \min\{1, v(s) + T(s)\} \qquad (1)$$

We can extend this definition to a sequence of requests in the obvious way, i.e. for $\sigma = T_1 T_2 \cdots T_k$, we let $v \wedge \sigma = (\cdots((v \wedge T_1) \wedge T_2) \cdots \wedge T_k)$. A phase ends when $v(s) \geq 1/2$ for all s. The task that causes a phase to end is considered part of that phase.

A phase will consist of one or more rounds. During a round, GREEDY remains in a single state and keeps track of a round cost g, which is set to 0 at the start of the round. For each task T requested during the round, we update g to g' by $g' = g + T(s)$, where s is the location occupied by GREEDY. GREEDY stays at s until a task T is requested which causes $g \geq 1$. T is served from s, then a new location s' is chosen for the next round. We say that GREEDY moves and a new round starts even if $s' = s$. Note that the last round of a phase may end prematurely (i.e. before $g \geq 1$).

We now describe how GREEDY chooses a new state for the next round. Intuitively, GREEDY will choose the state that will force the largest increase in the cost function during the next round. Define the *influence* of a task T on a cost function v to be the total increase in v on task T, i.e.,

$$\text{Influence}_v(T) = \sum_s ((v \wedge T)(s) - v(s)) \tag{2}$$

We can extend this definition to a sequence of tasks $\sigma = T_1 T_2 \cdots T_k$ by

$$\text{Influence}_v(\sigma) = \sum_{i=1}^{k} \text{Influence}_{v \wedge T_1 \cdots T_{i-1}}(T_i)$$

For $\sigma = T_1 T_2 \cdots T_k$, let $cost_\sigma(x) = \sum_{i=1}^{k} T_i(x)$ be the cost of σ on x. Let $\mathcal{S}_x$ be the set of task sequences which have cost at least 1 on x, i.e.,

$$\mathcal{S}_x = \{\sigma : cost_\sigma(x) \geq 1\}$$

We define the *influence* of a state x on a cost function v to be the minimum increase in v over task sequences from $\mathcal{S}_x$, i.e.,

$$\text{Influence}_v(x) = \min_{\sigma \in \mathcal{S}_x} \text{Influence}_v(\sigma) \tag{3}$$

Note that if GREEDY moves to x at the beginning of a round and the round does not end prematurely, then $\sum_s v(s)$ will increase by at least $\text{Influence}_v(x)$ during that round.

At the beginning of every round, we would like GREEDY to move to the location x that maximizes $\text{Influence}_v(x)$. Unfortunately, we do not know how to compute $\text{Influence}_v(x)$ in polynomial time. However, we can approximate $\text{Influence}_v(x)$. Specifically, we can compute a value $\mathrm{I}_v(x)$ in polynomial time where

$$\mathrm{I}_v(x) \leq \text{Influence}_v(x) \leq 2\,\mathrm{I}_v(x) \tag{4}$$

Thus, at the beginning of every round, GREEDY will move to the state x that maximizes $\mathrm{I}_v(x)$. We will see later exactly what $\mathrm{I}_v(x)$ is and how to compute it. For now it is enough to know that $\mathrm{I}_v(x)$ has many of the same properties as $\text{Influence}_v(x)$, including

Fact 3 *For any v_1 and v_2, if $v_1(s) \geq v_2(s)$ for all s, then $I_{v_2}(x) \geq I_{v_1}(x)$ for any x.*

Fact 4 *$I_v(x) \leq n$.*

Fact 5 *If $v(x) = d$, then $I_v(x) \geq 1 - d$*

Although GREEDY is not a strongly competitive algorithm for every uniform task system, it does approximate the best possible on-line algorithm. Specifically, we have:

Theorem 6 *The competitive ratio of GREEDY, $c_{GREEDY}(\mathcal{U}_n, \mathcal{T})$, is $O(\log n \cdot c(\mathcal{U}_n, \mathcal{T}))$.*

Proof. It is sufficient to show, for any completed phase, that $cost_{GREEDY}(phase)/cost_{opt}(phase)$ is $O(\log n \cdot c(\mathcal{U}_n, \mathcal{T}))$, where $cost_{opt}(phase)$ is a lower bound on the optimal cost during the phase, and $cost_{GREEDY}(phase)$ is an upper bound on GREEDY's cost. Fix any completed phase, and let k be the number of rounds in that phase. We first show

Claim 7 *The ratio $cost_{GREEDY}(phase)/cost_{opt}(phase)$ is $O(k)$.*

Proof. For each round, GREEDY incurs cost of at most 1 to move at the beginning of the round. During a round, the total cost to serve the requests, excluding the last request of the round, is at most 1. Let T_i be the last request of round i, and let $d_i = \min_x T_i(x)$. Then GREEDY incurs cost of at most $d_i + 2$ for the last request of round i. Letting $d = \sum_{i=1}^{k} d_i$, GREEDY's total cost for all k rounds is at most $d + 4k$. The optimal cost per phase is at least $\max\{1/2, d\}$. Thus, $\mathit{cost}_{GREEDY}(\mathit{phase})/\mathit{cost}_{opt}(\mathit{phase}) \leq (d+4k)/\max\{d, 1/2\} \leq 9k$. ■

Let s_i be the location of GREEDY during round i, and let v_i be the cost function at the beginning of round i. Consider the sequence $\mathrm{I}_{v_1}(s_1), \mathrm{I}_{v_2}(s_2), \ldots, \mathrm{I}_{v_k}(s_k)$. We know by fact 4 that $n \geq \mathrm{I}_{v_1}(s_1)$. By the definition of GREEDY, $\mathrm{I}_{v_i}(s_i) \geq \mathrm{I}_{v_i}(s_{i+1})$, thus by fact 3, we have $\mathrm{I}_{v_i}(s_i) \geq \mathrm{I}_{v_{i+1}}(s_{i+1})$ for each i. At the beginning of every round there must be a state x with $v(x) < 1/2$ (otherwise the phase ends), thus by fact 5, we must have $\mathrm{I}_{v_k}(s_k) > 1/2$. Thus we have

$$n \geq \mathrm{I}_{v_1}(s_1) \geq \mathrm{I}_{v_2}(s_2) \geq \cdots \geq \mathrm{I}_{v_k}(s_k) > 1/2$$

Now we divide the phase into subphases based on the $\mathrm{I}_{v_i}(s_i)$'s. Let subphase j consist of all rounds i where $(1/2)^{j-1} n \geq \mathrm{I}_{v_i}(s_i) > (1/2)^j n$. Note that there are at most $\lceil \log n + 1 \rceil$ subphases.

Let r be the maximum number of rounds in any subphase. Since k is $O(r \log n)$, the following claim implies our result.

Claim 8 *The maximum number of rounds r in any subphase is $O(c(\mathcal{U}_n, \mathcal{T}))$.*

Proof. We show, equivalently, that $c(\mathcal{U}_n, \mathcal{T})$ is $\Omega(r)$. We assume that $r \geq 4$ (the claim holds for $r \leq$ any constant).

Define the *total slack* of a cost function v by $\mathit{total\_slack}(v) = \sum_s (1 - v(s))$. Suppose subphase j has r rounds, $j_1, \ldots, j_r$. Note that the total slack decreases at round j_i by at least $\mathrm{I}_{v_{j_i}}(s_{j_i})$. Thus, we must have $\mathit{total\_slack}(v_{j_1}) \geq \sum_{i=1}^{r} \mathrm{I}_{v_{j_i}}(s_{j_i})$. Since $\mathrm{I}_{v_{j_i}}(s_{j_i}) > \mathrm{I}_{v_{j_1}}(s_{j_1})/2$ for each i, we have

$$\mathit{total\_slack}(v_{j_1}) > r\, \mathrm{I}_{v_{j_1}}(s_{j_1})/2 \tag{5}$$

From the choice of s_{j_1}, we know that for any x, $\mathrm{I}_{v_{j_1}}(x) \leq \mathrm{I}_{v_{j_1}}(s_{j_1})$, thus $\mathrm{Influence}_{v_{j_1}}(x) \leq 2\, \mathrm{I}_{v_{j_1}}(s_{j_1})$. Thus, for each x, there exists some $\sigma_x \in \mathcal{S}_x$ such that $\mathrm{Influence}_{v_{j_1}}(\sigma_x) \leq 2\, \mathrm{I}_{v_{j_1}}(s_{j_1})$. The influence can only decrease after more requests. Thus, for any σ, this implies

$$\mathrm{Influence}_{v_{j_1} \wedge \sigma}(\sigma_x) \leq 2\, \mathrm{I}_{v_{j_1}}(s_{j_1}) \tag{6}$$

Consider the following adversary which can be used against any on-line algorithm for $\langle \mathcal{U}_n, \mathcal{T} \rangle$. The adversary works in phases, and each phase consists of $\lfloor r/4 \rfloor$ rounds. For each round, if the algorithm is at x, the adversary requests the sequence σ_x until the algorithm moves or the entire sequence is requested. Note that in each round the on-line algorithm must incur a cost of at least 1 (either it stays at x or it moves).

We claim that for any phase there exists some state y which incurs cost less than 1 during the phase. To see this, suppose we set the cost function to v_{j_1} at the beginning of the phase. By inequality 6 the total slack decreases by at most $2\, \mathrm{I}_{v_{j_1}}(s_{j_1})$ per round, or at most $r\, \mathrm{I}_{v_{j_1}}(s_{j_1})/2$ per phase. By inequality 5 this is less than the total slack at the beginning of the phase, thus some slack must remain at the end of the phase. Thus, at the end of the phase some y must have $v(y) < 1$, and y must have incurred cost less than 1 during the phase. The adversary can move to such a y at the beginning of each phase and incur a total cost of less than 2 per phase. Thus, the competitive ratio $c(\mathcal{U}_n, \mathcal{T})$ is at least $\lfloor r/4 \rfloor / 2$, which is $\Omega(r)$. ■

It is easy to show that the additive constant b (from the definition of competitiveness) for GREEDY is $O(n)$. Since we have bounded $\mathit{cost}_{GREEDY}(\mathit{phase})/\mathit{cost}_{opt}(\mathit{phase})$ for any completed phase, we have just to bound the cost of GREEDY with respect to the optimal cost for the last, potentially incomplete, phase. First, note that there are at most $O(n)$ rounds in any phase of GREEDY, which follows from the fact that the total slack (as defined in claim 8) is n at the beginning of a phase and decreases by at least 1/2 during each round. Second, note that at any step during a round, GREEDY's cost for that round is at most 4 more than the optimal cost, which follows from the fact that GREEDY's cost for round i is at most $d_i + 4$ (where d_i is defined as in claim 7), and any algorithm, even the optimal off-line algorithm, will incur cost of at least d_i on the same step as GREEDY. Thus, at any time during a phase, GREEDY's cost will not exceed the optimal cost by more than 4 times the number of rounds, so b is $O(n)$.

In general, theorem 6 is the best possible result for GREEDY. There exist task systems $\langle \mathcal{U}_n, \mathcal{T} \rangle$ such that $c_{GREEDY}(\mathcal{U}_n, \mathcal{T}) = \Omega(\log n \cdot c(\mathcal{U}_n, \mathcal{T}))$. Consider the following 0-∞ task systems. For $k \geq 1$, let $n = 2^k - 1$ and let $\mathcal{U}_{2n+1}$ be the uniform metric space on the states $\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n, x\}$. For this example, we will represent a task by the set of states whose task cost is ∞. Let $T_a = \{a_1, \ldots, a_n\}$, $T_b = \{b_1, \ldots, b_n\}$, and $T_x = \{x\}$. For i a power of 2 between 2 and n, let $T_i = \{a_i, a_{i+1}, \ldots, a_{2i-1}, b_i, b_{i+1}, \ldots, b_{2i-1}\}$. For i between 2 and n, and i not a power of 2, let $T_i = \{a_i\}$. For i between 2 and n, let $S_i = \{b_i\}$. Let $\mathcal{T}_{2n+1}$ be the set of tasks $\{T_a, T_b, T_x, T_2, \ldots, T_n, S_2, \ldots, S_n\}$. It is straightforward to show that the optimal on-line algorithm for $\langle \mathcal{U}_{2n+1}, \mathcal{T}_{2n+1} \rangle$ moves only among the states a_1, b_1, and x, and achieves a competitive ratio of 2. The adversary can force a competitive ratio of $k + 1 = \Theta(\log n)$ for GREEDY as follows. Assume GREEDY starts in state x and the adversary requests T_x. Then the sequence of moves and requests continues as follows: $a_{2^{k-1}}, T_{2^{k-1}}, a_{2^{k-2}}, T_{2^{k-2}}, \ldots, a_2, T_2, a_1, T_a, b_1, T_b, x, T_x$. Note that these moves are perfectly consistent with the rules of GREEDY which moves to the state that maximizes the influence. Upon request T_x, a new round begins and the sequence is repeated.

It remains to define $\mathrm{I}_v(x)$ and to show that it can be computed easily. Recall that Influence$_v(x)$ was defined in terms of $\mathcal{S}_x$ (see equation 3). We start by providing an alternative definition of Influence$_v(x)$ (lemma 9) which we then modify slightly to define $\mathrm{I}_v(x)$.

Fix v and x for the remainder of this section. Let $\mathcal{T}_x$ be the set of tasks with non-zero cost on x, i.e.,

$$\mathcal{T}_x = \{T \in \mathcal{T} : T(x) > 0\}$$

Recall that a task T is simply a vector $(T(1), T(2), \ldots, T(n))$. Thus, we can create new "tasks" by taking linear combinations of these vectors. We define a set of tasks $\mathcal{F}_x$ corresponding to $\mathcal{S}_x$, except that we use $\mathcal{T}_x$ instead of $\mathcal{T}$ as our base set. $\mathcal{F}_x$ is defined as follows ($\mathbf{Z}$ denotes the set of integers).

$$\mathcal{F}_x = \{F = z_1 T_1 + \cdots + z_k T_k : z_i \in \mathbf{Z}, z_i \geq 1, T_i \in \mathcal{T}_x, F(x) \geq 1\}$$

Recall the definition of Influence$_v(F)$ from equation 2. We have

Lemma 9 *Influence$_v(x) = \min_{F \in \mathcal{F}_x}$ Influence$_v(F)$*

Proof. For any $F = z_1 T_1 + z_2 T_2 + \cdots + z_k T_k \in \mathcal{F}_x$, since F has integer coefficients, we can construct the task sequence $\sigma_F = (T_1)^{z_1} (T_2)^{z_2} \cdots (T_k)^{z_k}$ where $(T_i)^{z_i}$ denotes the task T_i repeated z_i times. Similarly, for any $\sigma = T_1 T_2 \cdots T_k \in \mathcal{S}_x$ we can construct the task $F_\sigma = T_1 + T_2 + \cdots + T_k$. Note that $F(y) = cost_{\sigma_F}(y)$ and $cost_\sigma(y) = F_\sigma(y)$ for any y, thus we must have Influence$_v(F) =$ Influence$_v(\sigma_F)$ and Influence$_v(\sigma) =$ Influence$_v(F_\sigma)$. Thus, the lemma follows from the fact that the sequence $\sigma = T_1 T_2 \cdots T_k$ that minimizes equation 3 can be chosen such that each $T_i \in \mathcal{T}_x$. ■

We define $\mathcal{G}_x$ similarly to $\mathcal{F}_x$, but allow real coefficients ($\mathbf{R}$ denotes the set of reals).

$$\mathcal{G}_x = \{F = r_1 T_1 + \cdots + r_k T_k : r_i \in \mathbf{R}, r_i \geq 1, T_i \in \mathcal{T}_x, F(x) \geq 1\}$$

$\mathrm{I}_v(x)$ is defined in terms of $\mathcal{G}_x$.

$$\mathrm{I}_v(x) = \min_{F \in \mathcal{G}_x} \mathrm{Influence}_v(F) \tag{7}$$

It is straightforward to verify facts 3, 4, and 5. We show inequality 4.

Lemma 10 *$I_v(x) \leq$ Influence$_v(x) \leq 2\, I_v(x)$*

Proof. The fact that $\mathrm{I}_v(x) \leq \mathrm{Influence}_v(x)$ follows from the fact that $\mathcal{F}_x \subseteq \mathcal{G}_x$. We need to show that $\mathrm{Influence}_v(x) \leq 2\,\mathrm{I}_v(x)$. Let $F = r_1T_1 + \cdots + r_kT_k$ minimize equation 7. Take $F' = \lceil r_1 \rceil T_1 + \cdots + \lceil r_k \rceil T_k$. Since each $r_i \geq 1$, we have $\lceil r_i \rceil \leq 2r_i$, thus $F'(y) \leq 2F(y)$ for any y. This implies $\mathrm{Influence}_v(F') \leq \mathrm{Influence}_v(2F) \leq 2\,\mathrm{Influence}_v(F) = 2\,\mathrm{I}_v(x)$. Since $F' \in \mathcal{F}_x$, $\mathrm{Influence}_v(x) \leq 2\,\mathrm{I}_v(x)$. ■

We need to show that $\mathrm{I}_v(x)$ can be computed easily. We show in lemma 11 that equation 7 is minimized for some F that is a multiple of a single task $T \in \mathcal{T}_x$. For any $T \in \mathcal{T}_x$, it is straightforward to show that of all tasks in $\mathcal{G}_x$ which are a multiple of T, the task which has the least influence is $F = \max\{1/T(x), 1\}\,T$. Thus, $\mathrm{I}_v(x)$ can be computed by computing the influence of $F = \max\{1/T(x), 1\}\,T$ for each $T \in \mathcal{T}_x$, which can be done in polynomial time.

Lemma 11 *For some $F = rT \in \mathcal{G}_x$, $I_v(x) =$ Influence$_v(F)$.*

Proof. It will be useful to have the following equality, which follows from equations 1 and 2.

$$\mathrm{Influence}_v(T) = \sum_s \min\{1 - v(s), T(s)\} \tag{8}$$

Let $F = r_1T_1 + \cdots + r_kT_k$ minimize equation 7. We have two cases to consider.

case 1: F includes some term r_iT_i with $r_iT_i(x) \geq 1$. We can assume that r_iT_i is the only term in F, since removing the other terms would not increase the influence of F, and we would still have $F \in \mathcal{G}_x$. Thus the lemma holds in this case.

case 2: Each term r_iT_i in F has $r_iT_i(x) < 1$. Let $\overline{T_i} = (1/T_i(x))\,T_i$ for each i. Note that $(1/T_i(x)) > r_i \geq 1$ and $\overline{T_i}(x) = 1$, so $\overline{T_i} \in \mathcal{G}_x$. We will show that $\mathrm{Influence}_v(F) \geq \mathrm{Influence}_v(\overline{T_j})$ for some j, which implies our result.

Let $\overline{r_i} = r_iT_i(x)$ and take $\overline{F} = \overline{r_1}\overline{T_1} + \cdots + \overline{r_k}\overline{T_k}$. We have $\overline{F}(s) = F(s)$ for each s, thus

$$\mathrm{Influence}_v(F) = \mathrm{Influence}_v(\overline{F}) \tag{9}$$

For each i, define T_i^* by $T_i^*(s) = \min\{1 - v(s), \overline{T_i}(s)\}$ and let $r_i^* = \overline{r_i}/\sum_j \overline{r_j}$. Take $F^* = r_1^*T_1^* + \cdots + r_k^*T_k^*$. Note that $T_i^*(s) \leq \overline{T_i}(s)$ for each s. Also note that $\sum_j \overline{r_j} = \sum_j r_jT_j(x) = F(x) \geq 1$, which implies that $r_i^* \leq \overline{r_i}$. Thus, $F^*(s) \leq \overline{F}(s)$ for each s, and

$$\mathrm{Influence}_v(\overline{F}) \geq \mathrm{Influence}_v(F^*) \tag{10}$$

Note that $F^*(s) = \sum_i r_i^*T_i^*(s) = \sum_i r_i^* \min\{1 - v(s), \overline{T_i}(s)\} \leq \sum_i r_i^*(1 - v(s)) = 1 - v(s)$ (where the last equality follows from the fact that $\sum_i r_i^* = 1$). Using this and equation 8, we have

$$\mathrm{Influence}_v(F^*) = \sum_s \min\{1 - v(s), F^*(s)\} = \sum_s F^*(s) \tag{11}$$

Choose j such that $\sum_s T_j^*(s)$ is minimized. We have

$$\sum_s F^*(s) = \sum_s \sum_i r_i^*T_i^*(s) = \sum_i r_i^* \sum_s T_i^*(s) \geq \sum_i r_i^* \sum_s T_j^*(s) = \sum_s T_j^*(s) \tag{12}$$

Using the definition of T_j^* we have

$$\sum_s T_j^*(s) = \sum_s \min\{1 - v(s), \overline{T_j}(s)\} = \text{Influence}_v(\overline{T_j}) \tag{13}$$

Combining inequalities 9–13, we have $\text{Influence}_v(F) \geq \text{Influence}_v(\overline{T_j})$, which is what we wanted to show. ■

3 The Complexity of Strongly Competitive Algorithms

In this section, we address the complexity of devising an on-line algorithm which achieves the smallest possible competitive ratio for a given task system. Our assumption here is that the distance matrix and all allowable tasks are part of the input. Thus, we measure the complexity of our problem as a function of the size of an explicit representation of the entire task system. We prove that even for the simplest of task systems, where the distance matrix is uniform and the tasks vectors either have 0 or ∞ entries, the problem of devising a strongly competitive algorithm is PSPACE-complete.

Let $L(Task)$ be the set of all $(\mathcal{U}, \mathcal{T}, B)$ such that there is a B-competitive on-line algorithm for the 0-∞ uniform task system $\langle \mathcal{U}, \mathcal{T} \rangle$. Throughout this section, a task is represented by the set of states whose value is ∞. We say that an algorithm A is c-competitive *with constant* b if for all σ,

$$cost_A(\sigma) \leq c \cdot cost(\sigma) + b.$$

We prove the following theorem:

Theorem 12 *$L(Task)$ is PSPACE-complete.*

We first prove that $L(Task)$ is PSPACE-hard and then prove that $L(Task)$ is in PSPACE. The proof that $L(Task)$ is PSPACE-hard is accomplished via a series of two log-space reductions over the following games:

$G(ST)$: In this game, the *state* player plays the role of the algorithm and the *task* player plays the role of the adversary. The input is a 0-∞ uniform task system $\langle \mathcal{U}, \mathcal{T} \rangle$. The game is played in a series of rounds where the state player seeks to minimize the number of rounds and the task player seeks to maximize the number of rounds. At the beginning of the game, all states are unmarked, and the game finishes when all states have been marked. During a round, the state player moves to an unmarked state, and the task player must choose a task that contains the state. All states in the chosen task become marked. We denote by $L(ST)$ all $(\mathcal{U}, \mathcal{T}, B)$ triplets such that the state player can force the game to finish in at most B rounds.

$G(QBF)$: The input is a set of clauses $\mathcal{C}$ over the variables $\{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$. The game is played in n rounds. In the i^{th} round, Player 1 assigns a value to x_i and then Player 2 assigns a value to y_i. At the end of the n rounds, if there is an unsatisfied clause, Player 1 wins, and if all the clauses are satisfied, Player 2 wins. $L(QBF)$ is the set of all sets of clauses for which Player 1 has a winning strategy.

$L(QBF)$ is well known to be PSPACE-complete ([14]).

Lemma 13 *$L(ST) \prec L(Task)$.*

Lemma 14 $L(QBF) \prec L(ST)$.

Proof of Lemma 13. Let $\langle \mathcal{U}, \mathcal{T} \rangle$ be an instance of $G(ST)$. If the state player can force the game $\langle \mathcal{U}, \mathcal{T} \rangle$ to finish in at most B rounds, we say that he has a B-strategy.

We use *work functions* which were first used in [4]. A work function maps the set of states onto the non-negative real numbers. Intuitively, the work function value on a state s at any point in time is the minimum cost of processing the sequence of tasks seen so far and ending up in state s. Thus, if s_0 is the initial state, then at the beginning of the sequence, $w(s_0) = 0$, and $w(s) = 1$ for all $s \neq s_0$. After each request T, w is updated as follows:

$$w_{new}(s) \leftarrow \min_{s'} w_{old}(s') + T(s') + d_{s,s'}$$

$w_{min} = \min_s w(s)$ is a lower bound on the optimal off-line cost.

We also use the notion of *marking* found in [7]. The task system is analyzed in phases where a new phase begins when w_{min} increases by one. Since the metric space is uniform and the tasks are all $0 - \infty$ vectors, $w(s)$ is either w_{min} or $w_{min} + 1$. We call a state whose work function value is $w_{min} + 1$ a *marked* state and a state whose work function value is w_{min} an *unmarked* state. A new phase begins just before a task is requested which includes all the unmarked states. At that point, all states become unmarked except those states that are contained in the currently requested task.

We construct a task system from $\langle \mathcal{U}, \mathcal{T} \rangle$ as follows. We add two states, a and b, to $\mathcal{U}$ to obtain $\mathcal{U}'$, and two tasks $\{a\}$ and $\{b\}$ to $\mathcal{T}$ to obtain $\mathcal{T}'$. We claim that $(\mathcal{U}, \mathcal{T}, B) \in L(ST)$ if and only if $(\mathcal{U}', \mathcal{T}', B+1) \in L(Task)$. First we show that if there is a B-strategy for $\langle \mathcal{U}, \mathcal{T} \rangle$, then there is a $B+1$ competitive algorithm for $\langle \mathcal{U}', \mathcal{T}' \rangle$. The algorithm alternates between phases where either a or b is the only marked state after the first request of the phase. If a phase starts with a marked, then the algorithm follows the B-strategy for $\langle \mathcal{U}, \mathcal{T} \rangle$. All the states in $\mathcal{U}$ are marked within B rounds. The last remaining unmarked state is b. The algorithm moves to b. Since the only task which contains b is $\{b\}$, a new phase will begin on the next request to $\{b\}$ after which only b is marked. The algorithm continues in this manner, only incurring a cost of $B+1$ per phase. There may be some initial cost on the first phase if a or b is not the initial state. However, this is a finite initial cost (at most n, where n is the number of states in $\mathcal{U}$) after which the algorithm can begin the second phase with a being the only marked state and will incur a cost of at most $B+1$ on all successive phases. The optimal algorithm incurs a cost of at least one per phase since $cost(\sigma) = \min_s w(s)$. Thus, the algorithm is $(B+1)$-competitive with constant n.

If $(\mathcal{U}', \mathcal{T}', B+1) \in L(Task)$, then there is an algorithm which is $(B+1)$-competitive with constant β, for some β. That is, if the algorithm plays against any adversary for at least $\beta + 1$ phases, there is a phase in which the algorithm incurs a cost of at most $B+1$. We will show how to use this algorithm to obtain a strategy for the state player for $\langle \mathcal{U}, \mathcal{T} \rangle$ which can play against any task player for $\beta + 1$ games such that at least one of these games will last for at most B rounds. Clearly, this would imply that $(\mathcal{U}, \mathcal{T}, B) \in L(ST)$. Otherwise, there is a task player which can force every game to take at least $B+1$ rounds. (We will show that for every phase in which the algorithm incurs a cost of X, there is a game which lasts $X - 1$ rounds).

A new state-task game begins whenever a new phase begins for the algorithm. Recall that $\mathcal{U}' = \mathcal{U} \cup \{a\} \cup \{b\}$, so the algorithm is always in a state in $\mathcal{U}$ or in a or b. If the algorithm is in a state $s \in \mathcal{U}$ at the beginning of a phase, then the state player goes to state s to start off the game. If the algorithm is in a or b at the beginning of the phase, then the state player waits until the first time the algorithm moves to an $s \in \mathcal{U}$ and starts the game off in s. The state player simulates the algorithm for $\langle \mathcal{U}', \mathcal{T}' \rangle$. Whenever the algorithm is on a (resp. b), the state player feeds the algorithm the task $\{a\}$ (resp. $\{b\}$). Whenever the task player chooses a task $T \in \mathcal{T}$, the state player feeds the corresponding task T to the algorithm. Whenever the algorithm moves to a state $s \in \mathcal{U}$, the state player moves to the corresponding state in the state-task game. Note that the set of marked states in $\mathcal{U}$ in the task system and the set of marked states in the state-task game is always the same. This continues until the task player requests a task T which contains every unmarked state in the state-task game, including the current state s occupied by the state player. We remark that the task player will only request such a task when every task which contains s contains all of the unmarked states. In this case, the task player can request any task which contains s and the game will end. We analyze three cases, depending on which nodes in $\{a, b\}$ are marked at this point:

Case 1: Both a and b are marked. The algorithm has incurred a cost of X while the state-task game has lasted $X - 2$ rounds (the extra cost of 2 comes from leaving states a and b during the phase). Both the state player and the algorithm are in some state $s \in \mathcal{U}$. The game ends after the next round (and thus will have lasted $X - 1$ rounds). The state player feeds the request T to the algorithm which becomes the first request of the next phase. The state player starts the next game in state s and T is the first request of the next game. Recall that since T can be any request that contains s, the task player is not restricted by forcing the last request of the previous game to be the first request of the next.

Case 2: One of a or b is unmarked (say it is a). The algorithm has incurred a cost of X while the state-task game has lasted $X - 1$ rounds. Since the only task which contains a is $\{a\}$, all states besides a are marked and the algorithm is in state a. Furthermore, the state-task game is over. The state player feeds the request $\{a\}$ to the algorithm. A new phase begins and a becomes the only marked state.

Case 3: Both a and b are unmarked. The algorithm has incurred a cost of X and the game has lasted X rounds. All states in $\mathcal{U}$ are marked and the algorithm is in either a or b (suppose it is a). The state player feeds the request $\{a\}$ to the algorithm. The algorithm incurs an additional cost of 1, moves to state b and then we are in the same situation as Case 2.

■

Proof of Lemma 14. In the first part of the proof, we reduce an instance $\mathcal{C}$ of $G(QBF)$ to an instance Φ of the game $G(MATCH)$ which we define below. Φ will not be a general instance of $G(MATCH)$ but will have a special form which will facilitate the second part of the proof in which we transform Φ into $(\mathcal{U}, \mathcal{T}, B)$, an instance of $G(ST)$. To avoid confusion, we rename the players for $G(QBF)$ Players A and B. Player A is trying to make $\mathcal{C}$ unsatisfied and Player B is trying to satisfy $\mathcal{C}$. The reduction will have the property that there is a winning strategy for Player A in $\mathcal{C}$ iff there is a winning strategy for Player 1 in Φ iff there is a B-strategy for the state player in $(\mathcal{U}, \mathcal{T})$. The first part of the reduction borrows heavily from techniques found in [11]. Here is the definition of $G(MATCH)$:

$G(MATCH)$: The input is any propositional formula Φ over two sets of variables V_1 and V_2. $|V_1| = |V_2|$ and each variable in V_1 is paired to a variable in V_2 via a matching π. In each round, Player 1 picks a variable $v \in V_1$ such that the value of v has not yet been determined. Player 1 assigns a value to v and Player 2 assigns a value to $\pi(v)$. At the end of the $|V_1|$ rounds, if Φ is unsatisfied, Player 1 wins and if Φ is satisfied, Player 2 wins. $L(MATCH)$ is the set of all formulae for which Player 1 has a winning strategy.

Part I:

The first reduction takes an arbitrary set of clauses over the variables $\{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ and produces a propositional formula Φ over $X \cup Y \cup U \cup W \cup S \cup T$ where

$$X = \{x_1, \ldots, x_n\}, \quad U = \{u_1, \ldots, u_n\}, \quad S = \{s_1, \ldots, s_n\}$$
$$Y = \{y_1, \ldots, y_n\}, \quad W = \{w_1, \ldots, w_n\}, \quad T = \{t_1, \ldots, t_n\}$$

Player 1's variables are $V_1 = (X \cup U \cup T)$ and Player 2's variables are $V_2 = (S \cup W \cup Y)$. The matching π is defined as follows: $\pi(x_i) = s_i$, $\pi(u_i) = w_i$, and $\pi(t_i) = y_i$.

The formula Φ will have the property that Player 2 can automatically win if any of the following happen:

1. Player 1 sets u_1 before s_1 has been set.
2. For any $i \in [2, \ldots, n]$, Player 1 sets u_i before y_{i-1} or s_i have been set.
3. For any $i \in [1, \ldots n]$, Player 1 sets t_i before w_i is set.

At the end of the game, if Player 1 has not disobeyed any of the above constraints, then Φ is satisfied if and only if the values of the variables in X and Y satisfy $\mathcal{C}$. The above conditions imply that for Player 1 to win, he must choose t_{i-1} before u_i (condition 2), and u_i before t_i (condition 3). Thus, the values of the y's are set in the following order: $y_1, y_2, \ldots, y_n$. Furthermore, Player 1 must also set x_i before u_i (condition 2), and hence before t_i (condition 3). Thus, the value of x_i is determined before the value of y_i. Since Player 1 wants to determine the values of each x_i as late as possible, he will choose to have the variables set in the QBF order: $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$. We construct the formula Φ as follows:

Let $\Phi_{\mathcal{C}}$ be the set of clauses $\mathcal{C}$ written as a formula in conjunctive normal form. Then Φ is chosen to be $\Phi_{\mathcal{C}} \vee \Phi'$ where:

$$\Phi' = \left(\bigvee_{1 \leq i \leq n} (w_i \equiv t_i) \right) \vee \left((u_1 \equiv s_1) \vee \bigvee_{1 < i \leq n} (u_i \equiv (s_i \oplus y_{i-1})) \right).$$

We claim that Player A has a winning strategy for $\mathcal{C}$ if and only if Player 1 has a winning strategy for Φ.

($\Longrightarrow$) Player 1 sets the variables in order from $i = 1$ up to n: x_i then u_i then t_i. Since w_i must be set before t_i, Player 1 can choose $t_i \leftarrow \bar{w}_i$. Thus, all the $(t_i \equiv w_i)$'s are false. Also the values of y_{i-1} and s_i are determined before Player 1 sets u_i, so Player 1 can set $u_i \leftarrow \neg(s_i \oplus y_{i-1})$. Thus, all the $(u_i \equiv (s_i \oplus y_{i-1}))$'s are false. When Player 1 determines the value of x_i, the values of variables $y_1, \ldots, y_{i-1}$ have already been determined, and Player 1 sets x_i according to the winning strategy for Player A. Since Player A's strategy is winning, $\Phi_{\mathcal{C}}$ also evaluates to false and the entire formula Φ is false.

($\Longleftarrow$) Suppose that Player B has a winning strategy for $\mathcal{C}$. We will show that Player 2 has a winning strategy for Φ. We explain Player 2's strategy for each variable type that he is forced to set:

w_i: If t_i is already set, then Player 2 sets $w_i \leftarrow t_i$. Then $(t_i \equiv w_i)$ is true and Φ is satisfied. If t_i has not been set, then w_i is set arbitrarily.

s_i: If u_i and y_{i-1} have been set, then Player 2 sets $s_i \leftarrow (u_i \oplus y_{i-1})$ (if $i = 1$, Player 2 sets $s_1 \leftarrow u_1$ if u_1 has been set). $(u_i \equiv (s_i \oplus y_{i-1}))$ (or $(u_1 \equiv s_1)$) evaluates to true and Φ is satisfied. Otherwise, s_i is set arbitrarily.

y_i: If u_{i+1} and s_{i+1} are already set, then Player 2 sets $y_i \leftarrow (u_{i+1} \oplus s_{i+1})$ and Φ will be satisfied. Otherwise, if $x_1, \ldots, x_i$ have already been set, then Player 2 sets y_i according to the strategy of Player B. If none of these conditions hold, then y_i is set arbitrarily.

The fact that Player 2 wins follows from the fact that either one of the precedent constraints are broken by Player 1 (in which case Player 2 automatically wins) or every time a y_i must be set, the values of $x_1, \ldots, x_i$ have already been determined and Player 2 wins by Player B's winning strategy.

Part II:

In the second part of the reduction, we produce an instance of $G(ST)$ from Φ such that there is a winning strategy for the state player if and only if there is a winning strategy for Player 1 in Φ. First we rewrite Φ' in disjunctive normal form:

$$\begin{aligned} \Phi' &= \bigvee_{1 \leq i \leq n} (w_i t_i \vee \bar{w}_i \bar{t}_i) \\ &\vee \ (u_1 s_1 \vee \bar{u}_1 \bar{s}_1) \\ &\vee \bigvee_{1 < i \leq n} (\bar{u}_i s_i y_{i-1} \vee \bar{u}_i \bar{s}_i \bar{y}_{i-1} \vee u_i s_i \bar{y}_{i-1} \vee u_i \bar{s}_i y_{i-1}) \end{aligned}$$

Φ can now be written as

$$\Phi = \bigwedge_{c \in \mathcal{C}} (c \vee \Phi').$$

We are now ready to describe $(\mathcal{U}, \mathcal{T})$. Let $|\mathcal{C}| = m$. Recall that $|V_1| = |V_2| = 3n$. There will be $m + 12n$ states in $\mathcal{U}$.

States:

- For each $v \in V_1$, we will have a state for each literal: $v, \bar{v}, \pi(v), \bar{\pi}(v)$. We call this set of states *group* v.
- For every clause $c \in \mathcal{C}$, there will be a state c. These states will be called the *clause* states.

Tasks:

There are five types of tasks:

Type 1: For any two clause states c and c', there is a task $\{c, c'\}$.

Type 2: For every clause c, and for every $v \in V_1$ such that v appears in c, there are two tasks $\{c, v, \pi(v)\}$ and $\{c, v, \bar{\pi}(v)\}$. Similarly, if $\bar{v}$ appears in c, there are two tasks, $\{c, \bar{v}, \pi(v)\}$ and $\{c, \bar{v}, \bar{\pi}(v)\}$. (See Figure 1).

Type 3: For every clause c, for every $\pi(v) \in V_2$ such that $\pi(v)$ appears in c, there is a task $\{c, \pi(v)\}$. Similarly, if $\bar{\pi}(v)$ appears in c, there is a task $\{c, \bar{\pi}(v)\}$. (See Figure 2).

Type 4: For each $v \in V_1$, we have the following four tasks:

$$\{v, \pi(v)\}, \{\bar{v}, \pi(v)\}, \{v, \bar{\pi}(v)\}, \{\bar{v}, \bar{\pi}(v)\}.$$

In addition, we will have tasks $\{\pi(v)\}$ and $\{\bar{\pi}(v)\}$. (See Figure 3).

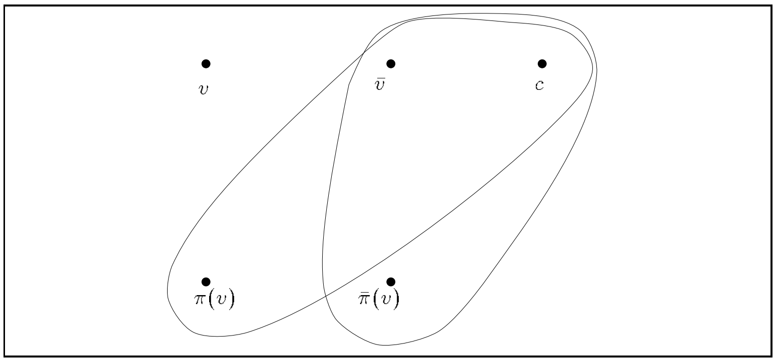

Figure 1: Type 2 tasks for $\bar{v} \in c$ and $v \in V_1$.

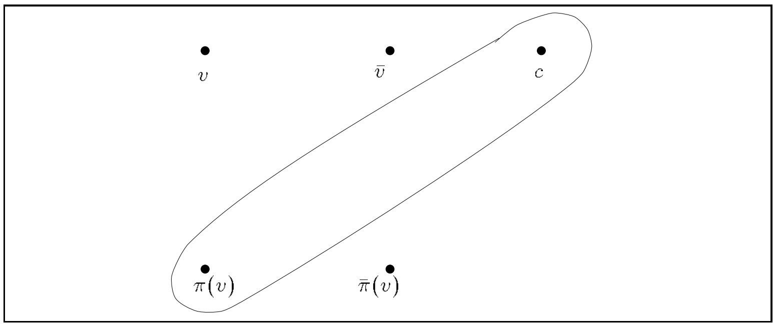

Figure 2: Type 3 tasks for $\pi(v) \in c$ and $\pi(v) \in V_2$.

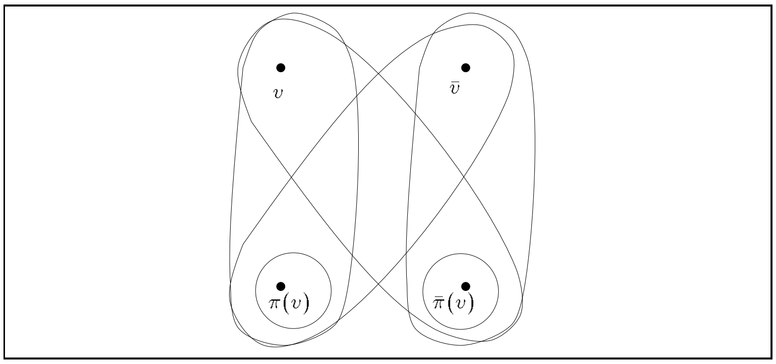

Figure 3: Type 4 tasks.

Type 5:

Before describing the type 5 tasks, we define the following families of subsets (which are not tasks). Each set is one of the conjuncts in Φ':

$$\mathcal{G}_1 = \{\{s_1, u_1\}, \{\bar{s}_1, \bar{u}_1\}\}.$$

For every $i \in [2, \ldots, n]$,

$$\mathcal{G}_i = \{\{s_i, y_{i-1}, \bar{u}_i\}, \{\bar{s}_i, \bar{y}_{i-1}, \bar{u}_i\}, \{s_i, \bar{y}_{i-1}, u_i\}, \{\bar{s}_i, y_{i-1}, u_i\}\}.$$

For every $i \in [1, \ldots, n]$,

$$\mathcal{F}_i = \{\{w_i, t_i\}, \{\bar{w}_i, \bar{t}_i\}\}.$$

Now we are ready to describe the type 5 tasks. For each $i \in [1, \ldots, n]$, for each set $g \in \mathcal{G}_i$, and for each clause c, we add the following two tasks to $\mathcal{T}$: $\{c\} \cup g \cup \{w_i\}$ and $\{c\} \cup g \cup \{\bar{w}_i\}$. Similarly, for each $i \in [1, \ldots, n]$, for each set $f \in \mathcal{F}_i$, and for each clause c, we add the following two tasks to $\mathcal{T}$: $\{c\} \cup f \cup \{y_i\}$ and $\{c\} \cup f \cup \{\bar{y}_i\}$.

A state s is said to be *dead* if it is unmarked and contained in a task such that the only other states in the task are marked. If the state player ever moves to a dead state s, the task player (who is trying to prolong the game) will request the task that only marks s. The game is said to be *dead* whenever every unmarked state is dead. As long as the game is not dead, the state player will never move to a dead state. Postponing such a move until the game becomes dead can only help the state player: it may already be marked later in the game. If it is not marked, the task player will choose the same (or equivalent) task as before.

Suppose that the game dies after k rounds and there are d marked vertices at the time it dies. Then when the entire game is over, it will have lasted $12n + m - d + k$ rounds. Thus, the state player would like to force $d - k$ to be large and the task player would like to force $d - k$ to be small. We claim that Player 1 has a winning strategy for Φ if and only if the state player can force $d - k$ to be at least $3n + 1$. (That is, the bound B is $m + 9n - 1$.)

Both players are said to play *legitimately*, if in the first $3n$ rounds, the state player chooses either v or $\bar{v}$ for some $v \in V_1$ such that none of $\{v, \bar{v}, \pi(v), \bar{\pi}(v)\}$ are marked. Suppose he picks v. In response to this, the task player picks either $\{v, \pi(v)\}$ or $\{v, \bar{\pi}(v)\}$, (i.e. a Type 4 task). A group v is said to be *set* as soon as $\pi(v)$ or $\bar{\pi}(v)$ is marked. Notice that as soon as a group is set, all of the states in the set are dead.

We first argue that if both players play legitimately, then Player 1 has a winning strategy if and only if the state player has a B-strategy (i.e. the state player can force $d - k \geq 3n + 1$). The vertices that are marked in the first $3n$ rounds correspond to the literals that are true. At the end of these $3n$ rounds, the truth assignment has been determined. The state player (playing according the optimal strategy for Player 1) has determined the values of the variables in V_1, and the task player (playing according to the optimal strategy for Player 2) has determined the values of the variables in V_2. All the groups have been set, and all the states in the groups are dead. If all the clause states are dead at this point, then after $3n$ rounds, $6n$ states are marked, the game is dead and $d - k \leq 3n$. On the other hand, if there is a clause state which is not dead, the state player can move there forcing two more states to be marked. This means that after $3n + 1$ moves, at least $6n + 2$ states are marked and $d - k \geq 3n + 1$. So we will prove that Φ is satisfied if and only if all clause states are dead.

Φ is satisfied. Either Φ' or $\Phi_\mathcal{C}$ is satisfied. If Φ' is satisfied, then there is a conjunct in Φ' such that every literal in the conjunct is satisfied. Let's say it corresponds to the set $g \in \mathcal{G}_i$. All the states in g are marked. Furthermore, either w_i or $\bar{w}_i$ is set to true (i.e. marked). Without loss of generality, suppose w_i is true. Then for every clause c, there is a Type 5 task, $\{c\} \cup \{w_i\} \cup g$. That is, every clause state is dead and the game is dead. If $\Phi_\mathcal{C}$ is satisfied, then every clause is satisfied. This means that every clause state is contained in some Type 2 or 3 task with only marked nodes, and hence is dead.

Φ is not satisfied. Both Φ' and $\Phi_\mathcal{C}$ are not satisfied. Since Φ' is not satisfied, every conjunction in Φ' contains an unsatisfied literal. Thus, every Type 5 task contains at least two unmarked states (a clause state and at least one literal state). Since $\Phi_\mathcal{C}$ is not satisfied, there is a clause c which contains only unsatisfied literals. Thus, every Type 2 or 3 task in which c is contained has at least two unmarked nodes (c and the unsatisfied literal). Thus, c is not dead.

It remains now to handle the cases where some player does not play legitimately. We must prove that:

(a) If the task player is the first to play illegitimately, then the state player can force $d - k \geq 3n + 1$.

(b) Suppose that the state player sets the variables according to the optimal strategy for Player 1, and the task player sets the variables according to the optimal strategy for Player 2. If Player 2 has a winning strategy and the state player is the first to play illegitimately, then the task player can force $d - k \leq 3n$.

Proof of (a). The only way for the task player not to play legitimately is if the state player is on state v or $\bar{v}$ in V_1 and the task player picks a task containing a clause state c, (i.e. a Type 2 or 5 task). Suppose the state player is on v. The chosen task also contains either $\pi(v)$ or $\bar{\pi}(v)$. In addition, it may contain some states in V_2 that cause a group to become set when the task is chosen. Call this set of groups R. The task marks at least $3+|R|$ states. The state player will then proceed to each unset group because all other states are dead. The task player will mark at least 2 states for each of the remaining unset groups. Thus, if the task player makes an illegitimate move, in $3n - |R|$ moves, at least $2(3n - |R| - 1) + 3 + |R|$ states will be marked, and $d - k \geq 3n + 1$.

Proof of (b). Suppose that the state player is the first to make an illegitimate move. The state player will never go to states that correspond to literals in V_2 until the game is dead. Furthermore, once $\pi(v)$ or $\bar{\pi}(v)$ is marked, the state player will never go to v or $\bar{v}$. Thus, an illegitimate move by the state player is one where he moves to some clause state c before all the groups have been set.

If the clause c contains a literal that is satisfied, then the state c is dead and we can assume the state player would not move to c. If c contains a literal v from V_2 whose value is not yet determined (i.e. the group containing it has not been set), then the task player picks $\{c, v\}$. Now all clause states are dead and one more group is set. The state player will move to unset groups until all groups are set. There are 2 states marked for every one of the n groups and the game is dead.

The last possibility is that for each literal in c, either the value of that literal is already determined to be false or it is a variable (or the negation of a variable) from V_1 whose value is not yet determined. We claim in this case that the variables have not been set up to this point according to a winning strategy for Player 2. We can assume that up to this point, the variables in V_1 have been set according to an optimal strategy for Player 1, and the variables in V_2 have been set according to an optimal strategy for Player 2. Thus, Player 1 (the state player) has set variables up to this point in an order which is consistent with the partial ordering enforced by Φ' (conditions 1-3). Player 1 can continue to set the variables in such a way that Φ' evaluates to false and can still set the remaining variables in X in any way he chooses. Since there is a clause c such that all the literals in c are either set to false or are unset variables from V_1 (i.e. from X), Player 1 can choose the setting of the remaining variables in c such that c evaluates to false. Thus $\Phi_{\mathcal{C}}$ and Φ' will both be false and Player 2 does not have a winning strategy. ∎

Now that we have shown that $L(Task)$ is PSPACE-hard, we give the proof that it is in PSPACE. We note that although we know that $L(Task)$ is in PSPACE, we have not been able to prove a similar result for task systems with arbitrary tasks or on non-uniform metric spaces.

Lemma 15 *$L(Task) \in PSPACE$.*

Proof. We first summarize a technique of Chrobak and Larmore ([6]) which, although it does not run in polynomial space, does give a decision procedure for $L(Task)$. Let $\langle \mathcal{M}, \mathcal{T} \rangle$ be a 0-∞ metrical task system. Chrobak and Larmore show that the existence of an on-line c-competitive strategy for $\langle \mathcal{M}, \mathcal{T} \rangle$ is equivalent to the existence of a so-called *c-potential function* for $\langle \mathcal{M}, \mathcal{T} \rangle$. For any finite $\langle \mathcal{M}, \mathcal{T} \rangle$ with rational distances and any rational c, there is a finite procedure for computing the minimum c-potential (if a c-potential exists). (Chrobak and Larmore assume integer values, but their results hold for rationals as well.) We describe this procedure when $\mathcal{M} = \mathcal{U}_n$.

Let $\mathcal{W}$ be the set of integer-valued work functions on $\mathcal{U}_n$. Since the metric space is uniform and all tasks are 0-∞ tasks, all "reachable" work functions are integer-valued. For any work function w there is a corresponding *offset* function w' defined by $w' = w - \inf w$ (where $\inf w$ is shorthand for $\inf_y w(y)$). w' is called an offset function since $\inf w' = 0$. Let $\mathcal{W}^0$ be the set of offset functions corresponding to $\mathcal{W}$. Note that $|\mathcal{W}^0| = 2^n - 1$ since it includes all 0-1-valued functions except the all 1's function. For any work function w and task T, we let $w \wedge T$ denote the work function that results from requesting task T when w is the current work function, and we let $w \bigtriangleup T$ denote the offset function corresponding to $w \wedge T$.

We call a pair $(w, x) \in \mathcal{W} \times \mathcal{U}_n$ a *configuration*. To compute the minimum c-potential, it is sufficient to consider only configurations (w, x) where $w \in \mathcal{W}^0$ (see [6] for details). A *potential function* ϕ is a function from configurations to $\mathbf{R}^+$, where $\mathbf{R}^+$ is the set of non-negative reals. By $\phi_1 \leq \phi_2$, we mean $\phi_1(w, x) \leq \phi_2(w, x)$ for all w and x. We say that a potential function ϕ is a *c-potential* if for any configuration $(w, x) \in \mathcal{W}^0 \times \mathcal{U}_n$ and any task T, there exists a $y \notin T$ such that,

$$\phi(w, x) \geq \phi(w \bigtriangleup T, y) + d_{x,y} - c \cdot \inf(w \wedge T) \tag{14}$$

We can compute the minimum c-potential for $\langle \mathcal{U}_n, \mathcal{T} \rangle$, if it exists, as follows. We compute a sequence of potentials $\phi^0, \phi^1, \phi^2, \ldots$. For each ϕ^i, we compute $\phi^i(w, x)$ for all configurations $(w, x) \in \mathcal{W}^0 \times \mathcal{U}_n$. We let $\phi^0(w, x) = 0$ and compute ϕ^{i+1} from ϕ^i as follows:

$$\phi^{i+1}(w, x) = \max_T \min_{y \notin T} \{\phi^i(w \bigtriangleup T, y) + d_{x,y} - c \cdot \inf(w \wedge T)\} \tag{15}$$

Note that $\phi^{i+1} \geq \phi^i$ for all i. We stop when one of two things happens, either $\phi^{i+1} = \phi^i$, in which case ϕ^i is a c-potential, or $\phi^{i+1} > 0$ everywhere, which implies that the sequence diverges and a c-potential does not exist. Because there are a finite number of configurations and c is rational, one of these two events happens within a finite number of iterations. Thus, this procedure is a decision procedure for $L(Task)$. Unfortunately, there are exponentially many configurations, so this procedure does not run in polynomial space.

We modify the above procedure to run in polynomial space. We compute a possibly different sequence of functions $\psi^0, \psi^1, \psi^2, \ldots$. For each function, instead of storing the entire function at each step, we save the values $\psi^i(w,x)$ for a subset of the configurations, and compute the rest of the values as we need them. For each $T \in \mathcal{T}$, let w_T be the offset function defined by $w_T(x) = 0$ if $x \notin T$ and $w_T(x) = 1$ if $x \in T$. Let $\mathcal{W}^0_{\mathcal{T}} = \{w_T : T \in \mathcal{T}\}$ be the set of all such offset functions. The intuition for this definition is the following: if we analyze a uniform task system in terms of phases, where each new phase starts when all states in the previous phase are marked, then $\mathcal{W}^0_{\mathcal{T}}$ is the set of offset functions that can start a phase. Suppose we have a function ψ which has been defined for all $(w,x) \in \mathcal{W}^0_{\mathcal{T}} \times \mathcal{U}_n$. We can extend ψ to all $(w,x) \in \mathcal{W}^0 \times \mathcal{U}_n$ by defining, for $w \notin \mathcal{W}^0_{\mathcal{T}}$,

$$\psi(w,x) = \max_{\substack{T \\ w \wedge T \neq w}} \min_{y \notin T} \{\psi(w \bigtriangleup T, y) + d_{x,y} - c \cdot \inf(w \wedge T)\} \tag{16}$$

where if $w \bigtriangleup T \notin \mathcal{W}^0_{\mathcal{T}}$ the value for $\psi(w \bigtriangleup T, y)$ is defined recursively.

Claim 16 *Given $\psi(w,x)$ for all $w \in \mathcal{W}^0_{\mathcal{T}}$, the values $\psi(w,x)$ for $w \notin \mathcal{W}^0_{\mathcal{T}}$ are well-defined, and furthermore $\psi(w,x)$ can be computed in polynomial space.*

Proof. We first show that ψ is well-defined. We prove this by induction on the number of zeros in w (i.e., the number of states x where $w(x) = 0$). If w has only one zero, then for any task T for which $w \wedge T \neq w$, it will be the case that $w \bigtriangleup T = w_T \in \mathcal{W}^0_{\mathcal{T}}$, thus $\psi(w,x)$ is well-defined for all w with at most one zero. Suppose ψ is well-defined for any (w', x) where w' has at most $k-1$ zeros, and suppose w has k zeros. Then for any task T where $w \wedge T \neq w$, $w \bigtriangleup T$ will either have fewer zeros than w or will be in $\mathcal{W}^0_{\mathcal{T}}$. In either case, $\psi(w \bigtriangleup T, y)$ is well-defined for each y, and $\psi(w,x)$ is well-defined for each x.

To see that $\psi(w,x)$ can be computed in polynomial space, observe that the depth of the recursion to compute $\psi(w,x)$ is at most the number of zeros in w, which is at most n. ∎

We compute a sequence of functions $\psi^0, \psi^1, \ldots$. To "compute" ψ^i, we merely compute and save the values $\psi^i(w,x)$ for $(w,x) \in \mathcal{W}^0_{\mathcal{T}} \times \mathcal{U}_n$, which is polynomially many values. (The values $\psi^i(w,x)$ for $w \notin \mathcal{W}^0_{\mathcal{T}}$ are defined by equation 16, however, these values are not saved.) The sequence $\psi^0, \psi^1, \ldots$ is defined as follows: ψ^0 is defined by setting $\psi^0(w,x) = 0$ for all $(w,x) \in \mathcal{W}^0_{\mathcal{T}} \times \mathcal{U}_n$. Given ψ^i, ψ^{i+1} is computed as in equation 15 (substituting ψ for ϕ in the equation) for each $(w,x) \in \mathcal{W}^0_{\mathcal{T}} \times \mathcal{U}_n$. To do this, we may need values $\psi^i(w,x)$ for $w \notin \mathcal{W}^0_{\mathcal{T}}$, but by claim 16, we can compute each such value in polynomial space. Thus, computing ψ^{i+1} from ψ^i requires only polynomial space.

Note that $\psi^{i+1} \geq \psi^i$ for all i. We compute the sequence until either $\psi^{i+1} = \psi^i$ for some i or $\psi^{i+1}(w,x) > 0$ for all $(w,x) \in \mathcal{W}^0_{\mathcal{T}} \times \mathcal{U}_n$. Since there are a finite number of configurations and c is rational, one of these two events happens within a finite number of iterations. Suppose the sequence diverges. Observe that $\psi^i \leq \phi^i$ for every i (this follows by a simple induction). Thus the sequence $\phi^0, \phi^1, \phi^2, \ldots$ must also diverge, and a c-potential cannot exist. Suppose the sequence converges. We note that when ψ^i is extended to all (w,x) using equation 16, $\psi^i(w,x)$ may be negative for some $(w,x) \notin \mathcal{W}^0_{\mathcal{T}} \times \mathcal{U}_n$. Thus, ψ^i may not be a potential function. However, ψ^i must satisfy equation 14 (otherwise ψ^{i+1} would not equal ψ^i), so $\psi^i - \inf \psi^i$ is a c-potential. Thus, we have a polynomial space decision procedure for $L(Task)$.

We note that our procedure can be simplified. Instead of defining a potential for each configuration (w,x), we really only need to define a potential for each offset function w. Intuitively, we can do this since for any offset function w, there will be an "optimal" state x to be in, and, since the metric space is uniform, we can assume that whenever the on-line algorithm moves, it moves to the optimal state for the new work function. We omit the details. ∎

The reductions of Lemmas 13 and 14 show that the problem of computing $c(\mathcal{U}, \mathcal{T})$ is PSPACE-hard. However, this does not immediately imply that devising a strongly competitive on-line strategy for a given task system is PSPACE-hard. Perhaps an algorithm exists which when given any task system $\langle \mathcal{M}, \mathcal{T} \rangle$ and any sequence σ of tasks in $\mathcal{T}$ on-line, uses polynomial time per request and is guaranteed to be $c(\mathcal{M}, \mathcal{T})$-competitive. However, if we insist that the algorithm be $c(\mathcal{M}, \mathcal{T})$-competitive with additive constant bounded by a polynomial in n and d (where n is the number of states and d is the largest distance in $\mathcal{M}$), then we can show that the existence of such an algorithm implies that P=PSPACE. We prove the following theorem:

Theorem 17 *Suppose there is a polynomial q and an algorithm which when given any task system $\langle \mathcal{M}, \mathcal{T} \rangle$ and any sequence σ of tasks in $\mathcal{T}$ on-line, uses polynomial time per request and is $c(\mathcal{M}, \mathcal{T})$-competitive with constant $q(\max\{n, d\})$ (where n is the number of states and d is the maximum distance in $\mathcal{M}$), then P=PSPACE.*

Proof. It it sufficient to show the result for 0-∞ uniform task systems, in which case the additive constant is simply $q(n)$. Using the reduction in Lemma 13, we can take any instance $\langle \mathcal{U}, \mathcal{T} \rangle$ of $G(ST)$ (where $\mathcal{U}$ has $n-2$ states) and construct a $0-\infty$, uniform task system $\langle \mathcal{U}', \mathcal{T}' \rangle$ (where $\mathcal{U}'$ has n states) such that $(\mathcal{U}, \mathcal{T}, B) \in L(ST)$ if and only if $\langle \mathcal{U}', \mathcal{T}', B+1 \rangle \in L(Task)$. Furthermore, the reduction has the property that we can use a polynomial time algorithm for $\langle \mathcal{U}', \mathcal{T}' \rangle$ which is $(B+1)$-competitive with constant $q(n)$ (if one exists) to derive a polynomial time strategy for the state player which can play any task player $(2q(n)+1)$ times such that the total number of rounds in all games is at most $B(2q(n)+1)+q(n) = (B+1)q(n)+B(q(n)+1)$.

Using the reduction in Lemma 14, we can take an arbitrary instance $\mathcal{C}$ of $G(QBF)$ and obtain an instance $\langle \mathcal{U}, \mathcal{T} \rangle$ of $G(ST)$ and a value B such that there is a winning strategy for Player 1 for $\mathcal{C}$ if and only if $(\mathcal{U}, \mathcal{T}, B) \in L(ST)$. The state-task game $\langle \mathcal{U}, \mathcal{T} \rangle$ produced in the reduction has the property that the task player can always force the game to last for at least B rounds. We call a task player *good* if he plays in such a way that every game against any state player will last at least B rounds. Let the number of states in $\mathcal{U}$ be $n-2$. If there is a state player which can play against any task player $2q(n)+1$ times such that the total number of rounds is at most $(B+1)q(n)+B(q(n)+1)$, then this state player can play against any good task player $2q(n)+1$ times such that at least $q(n)+1$ games last for at most B rounds. Thus, we can use the reduction in Lemma 14 and the polynomial time algorithm for the state player to derive a polynomial time algorithm for Player 1 in $G(QBF)$ which when given any set of clauses $\mathcal{C} \in L(QBF)$ can play against any Player 2 $(2q(n)+1)$ times and win at least a majority of the time.

Now consider a version of $G(QBF)$ where the input is not necessarily a set of clauses but an arbitrary boolean formula Φ. Call this game $G(BOOL)$ and the set of inputs for which Player 1 has a winning strategy $L(BOOL)$. Using standard techniques, $G(BOOL)$ can be reduced to $G(QBF)$. The reduction implies that if there is a polynomial time strategy which takes as input an any set of clauses in $L(QBF)$ and always wins at least one $q(n)+1$ out of $2q(n)+1$ times against any Player 2, then there is a polynomial time algorithm for Player 1 for any $\Phi \in G(BOOL)$ which will win at least $q(n)+1$ games out $2q(n)+1$ against any Player 2. We will show that the existence of such an algorithm can be used to obtain a polynomial time algorithm to decide $L(BOOL)$ which in turn implies that P=PSPACE.

Consider some $\Phi \notin L(BOOL)$. A winning strategy for Player 1 on $\neg\Phi$ is exactly a winning strategy for Player 2 on Φ. (To maintain the convention of $G(BOOL)$ that the universal player plays first, a dummy variable must be added to the beginning and end of $\neg\Phi$ and the variables renamed). Thus, we can use our algorithm for Player 1 to obtain an algorithm for Player 2 which takes as input any boolean formula $\Phi \notin L(BOOL)$ and plays against any Player 1 $(2q(n)+1)$ times and wins at least $(q(n)+1)$ times. For any formula Φ, we can determine in polynomial time if $\Phi \in L(BOOL)$: use the two algorithms to simulate Players 1 and 2 on Φ $(2q(n)+1)$ times. If Player 1 wins at least $(q(n)+1)$ times, then $\Phi \in L(BOOL)$. If Player 2 wins at least $(q(n)+1)$ times, then $\Phi \notin L(BOOL)$. ■

4 Extensions and future directions

4.1 Non-uniform metric spaces

We can extend GREEDY to any finite metric space by ignoring the actual distances and pretending that all distances are the same as the shortest distance in the space (in fact, this can be done with any algorithm designed for uniform metric spaces). If D is the ratio of the longest to shortest distance in the metric space, then this gives an approximation factor of $O(D \log n)$. We can apply this result to the paging problem described in the introduction. In this case, the ratio D is simply k. Thus, GREEDY achieves a competitive ratio for paging of, at worst, $O(k \log \binom{p}{k} \cdot c(\mathcal{U}_n, \mathcal{T})) = O(k^3 \log p)$. Using a different analysis, we can actually show that GREEDY is almost optimal for paging. (In fact, with only slight modifications, GREEDY is optimal.) This follows from the fact that, for 0-∞ tasks, GREEDY is "almost" a marking algorithm (see the proof of Lemma 13).

Consider a "node-weighted" task system where each state has a weight and the cost of travelling to a state is its weight, and let D be the ratio of the maximum to the minimum weight. We can extend GREEDY for this special case and show that GREEDY achieves an approximation factor of $O(\log D \log n)$.

4.2 Open Problems

The next step in this research is to improve the approximation factor by minimizing its dependence on the metric space. A significant step would be an algorithm for an arbitrary metric space where this factor has only logarithmic dependence on n and D. Similarly, we could ask if the $\log n$ factor could be eliminated from the approximation factor for uniform metric spaces. The ultimate goal would be an algorithm for any metric space whose approximation factor does not depend on the metric space. Alternatively, is it PSPACE-complete to even approximate the best competitive ratio by better than a logarithmic factor?

An interesting feature of our universal algorithm is that its competitive ratio is guaranteed to to be within an $O(\log n)$ factor of the best possible on-line algorithm, but it gives you no idea of what the best competitive ratio is. In order to determine this approximation, we would have to play GREEDY against the optimal adversary which we do not necessarily know. Is there a way of approximating $c(\mathcal{M}, \mathcal{T})$ given $\langle \mathcal{M}, \mathcal{T} \rangle$?

A less defined but very intriguing direction to investigate is the question of whether there is a natural and useful class of task systems for which the $O(\log n)$ factor in the approximation can be removed. This question is based on the observation that all of the examples of on-line problems which are special cases of task systems have task sets $(\mathcal{T})$ which can be described more efficiently than simply enumerating all tasks in $\mathcal{T}$. Can this regularity be exploited? Is it possible to define a property that many of these problems share and devise a universal algorithm which is $O(c(\mathcal{M}, \mathcal{T}))$-competitive for any task system $\langle \mathcal{M}, \mathcal{T} \rangle$ with the property?

References

[1] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. of the 24th Symposium on Theory of Computation*, pages 39–48, 1992.

[2] D.L. Black, and D.D. Sleator. Competitive algorithms for replication and migration problems. Technical report CMU-CS-89-201, Carnegie-Mellon University, 1989.

[3] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive Paging with Locality of Reference. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 249–259, 1991.

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

[5] M. Chrobak and L. Larmore. The server problem and on-line games. In *On-Line algorithms: proceedings of a DIMACS workshop. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:11–64, 1992.

[6] M. Chrobak and L. Larmore. Metrical service systems: deterministic strategies. Technical report UCR-CS-93-1, Dept. of Computer Science, University of California, Riverside, 1993.

[7] A. R. Karlin, M. S. Manasse, L. Rudolph, and D.D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):70–119, 1988.

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

[9] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. of the 20th Ann. ACM Symp. on Theory of Computing*, pages 322–333, May 1988.

[10] C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[11] T. Schaefer. On the Complexity of Some Two-person Perfect-Information Games. *Journal of Computer and System Sciences* (1), pages 185 – 225 (1978).

[12] D.D. Sleator, R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, Vol. 28, pages 202–208, February 1985.

[13] D.D. Sleator, R.E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, Vol. 32, No. 3, pages 652–686, July 1985.

[14] L. J. Stockmeyer, A. R. Meyer. Word problems requiring exponential time. In *Fifth Annual Symposium on the Theory of Computing*, pages 1 – 9, (1973).

[15] J. Westbrook. Randomized algorithms for multiprocessor page migration. *SIAM Journal on Computing*, Vol. 23, No. 5, pages 951–965, October 1994.