The Dynamic Servers Problem

Moses Charikar* Dan Halperin† Rajeev Motwani‡

Abstract

We present a generalization of the k-server problem, called *dynamic servers*, where the number of servers is not fixed a priori; rather, the algorithm is free to increase and decrease the number of servers at will, but it is required to pay a rental cost for each active server at designated times. This new problem is a simultaneous abstraction for problems arising in a variety of applications, particularly the information delivery problem for video-on-demand, web server management, and the problem of dynamic maintenance of kinematic structures for applications in molecular biology, simulation of hyperredundant robots, collision detection, and computer animation. The problem also appears to be of theoretical significance as a natural new paradigm in the realm of online algorithms. We give approximation algorithms for the offline problem, and initiate the study of the online version of this problem. We present an $O(\min\{\log n, \log \rho\})$-competitive algorithm where n is the number of requests and ρ is a parameter that depends on the metric space. We also prove a lower bound of $\Omega(\frac{\log\log\rho}{\log\log\log\rho})$ on the competitive ratio of any online algorithm for this problem. Our results are based on a geometric reformulation of the dynamic servers problem that leads to interesting connections with Steiner trees and geometric partitioning problems, and our results may be of independent interest in that context.

*Department of Computer Science, Stanford University, Stanford, CA 94305-9045. E-mail: `moses@cs.stanford.edu`. Supported by an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

†Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: `danha@math.tau.ac.il`. Supported in part by an Alon Fellowship, by ESPRIT IV LTR Project No. 21957 (CGAL), by the USA-Israel Binational Science Foundation, by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities, and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

‡Department of Computer Science, Stanford University, Stanford, CA 94305-2140. E-mail: `rajeev@cs.stanford.edu`. Supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

1 Introduction

We introduce the *dynamic servers* problem, a generalization of the k-server problem [15, 21]. This problem is a simultaneous abstraction for problems arising in a variety of applications described below and appears to be of theoretical significance as a natural new paradigm in online algorithms.

The *k-server* problem is the following: coordinate the movement of k mobile servers in a metric space, so as to process a sequence of requests at the points of the metric space, where a request is processed by moving a server to its location. The goal is to minimize the total distance traveled by the servers. In the online setting, the performance of an algorithm is measured via the *competitive ratio* [15, 26]. Culminating intensive research [15], the recent celebrated result of Koutsoupias and Papadimitriou [18] gives a $(2k-1)$-competitive k-server algorithm. Informally, the *dynamic servers* problem is a generalization of the k-server problem where the number of servers is not fixed at k; instead, the number of servers can be increased or decreased at will, the only catch being that the algorithm has to pay *rental* cost proportional to the number of servers active during any given period. We consider two possible models — one where the algorithm pays for deleting servers and another where deleting servers is free. Both of these are relevant to different applications as described below.

We describe the formal model for dynamic servers in Section 2. In Section 3, we consider the product of the metric space with the time axis. Placing the request points in the product space, the solution to the offline dynamic servers problem turns into a variant of the rectilinear Steiner arborescence problem studied earlier by Rao, Sadayappan, Hwang, and Shor [25]. In Section 4, we point out connections to past work on Steiner trees [5] and exploit the geometric reformulation to obtain an $O(\min\{\log n, \log \rho\})$-approximation to the offline problem with n requests in a metric space, where ρ denotes the ratio of the maximum to the minimum distance amongst the requested points. In Section 5, we present an online algorithm with competitive ratio $O(\min\{\log n, \log \rho\})$. We emphasize that the competitive ratio is bounded by a constant which depends on the (normalized) diameter of the metric space; our lower bound shows that such dependence is inevitable. In Section 6, we establish a lower bound of $\Omega(\frac{\log\log\rho}{\log\log\log\rho})$ on the competitive ratio, even when the metric space is merely the discrete line. Finally, in Section 7, we present a 6-approximation algorithm for the offline problem on the line, and also an improvement for the video-on-demand application (discussed below).

Several interesting questions remain open. Does randomization help for this problem? Then there is the question of improving both our offline and online bounds. Also, no online results are known for the extension where rental costs depend on time and server location.

Motivation and Applications.

One application of our model is to the information delivery problem for video-on-demand of Papadimitriou, Ramanathan, and Rangan [22, 23, 24]. For home entertainment channels, it is required to deliver personalized video programs across communication networks [20]. This involves the delivery across networks of the same data at many different sites and different times. The data is cached or replicated at intermediate sites, to minimize the cost of communication. The replication sites may adapt to customer requests. In the dynamic servers abstraction, the network is the metric space, the replication sites are the servers, customers issue requests, the replication storage cost is the rental cost, and the communication cost is the server movement cost. One twist in this application is that the rental cost of a server may depend on its location. Papadimitriou et al [24] demonstrate the NP-hardness of their problem, connect it to Steiner trees [5], and thereby provide a 2-competitive online algorithm for the special case of paths when customer requests appear only at one endpoint.

A related application is to managing a network of web servers under a single provider, as for example in the RADAR project (Replicators And Distributors and Redirectors) at AT&T Research [27]. Consider the web servers to be nodes in a metric space with distances determined by the network topology. Web objects (i.e., pages) can be stored at one or more nodes. Nodes receive requests for objects and service them by obtaining the object from the nearest node where it is stored. An object can be replicated from one node to another. We also pay a storage cost for storing an object at a node. The objective is to optimize accesses to web objects by replicating objects intelligently in response to the stream of requests. The problem can be modeled as a dynamic server problem (without deletion costs) for each object.

Our motivating application is the design of data structures and algorithms for the maintainence of kinematic structures, as described by Halperin, Latombe, and Motwani [12]. Such structures are used to maintain conformations of molecules in computational biology [9, 12, 16], motion planning in robotics [6, 10, 28] and computer animation [17]. Consider a collection of rigid bodies moving in 3-dimensional space and hinged together in a kinematic structure. We model these objects as a graph (typically, a path or a tree) whose vertices are the rigid bodies and edges are the linkages. The problem is to maintain data structures that capture the 3-dimensional shape as it evolves, and that support operations such as range queries or queries concerning possible self-collisions. Halperin, Latombe, and Motwani [12] proposed a model for this family of problems which is essentially the dynamic server problem, although their results were only applicable to the offline setting.

Previous Work.

In the offline setting, Halperin, Latombe, and Motwani [12] showed that even in the case of paths, devising the best strategy is NP-hard. They also presented an $O(\log n)$-approximation algorithm for paths, where n is the total number of updates. They point out the importance of the online version in the applications, and posed the open problem of devising an efficient online algorithm for minimizing the total cost over a request sequence. The special case of dynamic servers *without deletion costs* and *on the line* turns out to be the same as the Online Symmetric Rectilinear Arborescence problem studied by Berman and Coulston [4]. For this very special case of the general dynamic server problem, they gave an $O(\log n)$ competitive algorithm.

Our model may appear similar to the distributed file allocation problem [2, 3] where multiple copies of a file are be maintained at the nodes of a network so as to minimize communication costs for read/write requests. Copies may be replicated or discarded at will. There is a transmission cost for replication, but deletion is free. Moreover, there is no notion of a rental cost; instead, there is a write cost which is not linear in the number of copies present. These results do not appear to apply to our problem.

2 The Dynamic Servers Problem

The *dynamic servers* problem is defined as follows. It is required to maintain a dynamic collection of servers at the points of a possibly infinite metric space $\mathcal{M}$. The goal is to efficiently process a sequence of requests which are points in $\mathcal{M}$, where a request is serviced by moving a server to its location at cost equal to the distance moved. It is possible to create and destroy servers in a manner to be described shortly. The requests arrive at integer times $t \in \{1, 2, 3, \ldots\}$, and the set of requests received at time t is called *batch* t. At time t, the batch t requests must be serviced *online* in order of arrival. After the batch t requests have been serviced, the algorithm is charged *rental* cost equal to the number of currently active servers. We denote the overall number of requests by n and the total number of batches by m.

The following primitive operations can be performed on the servers: *move* a server from a point u to a point v at a cost equal to the distance $d(u, v)$; *clone* a copy of a currently active server at the same location, without incurring any cost except subsequent rental payments; and, *merge* two servers present at the same location into a single server at that location without incurring any cost. Combining these operations gives non-primitive operations, notably the *creation* and *deletion* of servers at arbitrary points. The cost of creating or deleting a server at any location is the distance to a nearest distinct server.

The process unfolds as follows. The initial state consists of a single server at some specified point in $\mathcal{M}$ and no rental cost is paid at time 0. Clearly, at least one server must be present at all times, since servers can only be destroyed by merging them into other servers. At time t, batch t requests start arriving and are processed in order of arrival, possibly by increasing the number of servers. Having serviced the requests in batch t, the algorithm may delete servers to save on rental costs. Finally, the algorithm pays rental cost equal to the number of currently active servers. In the time interval $(t, t+1)$, the configuration of the algorithm does not change, i.e., no servers are moved, created, or deleted. The rental cost paid by the algorithm can be viewed as the cost of maintaining its servers from time t to $t + 1$. At time $t + 1$, the entire process repeats.

Let Δ and δ denote the maximum and minimum distances, respectively, between the points of $\mathcal{M}$, and define the *normalized diameter* of $\mathcal{M}$ as $\rho = \Delta/\delta$. Our performance bounds will be in terms of ρ; when the entire metric space is not spanned by the request sequence, we assume that ρ refers to the subspace of $\mathcal{M}$ *presented in the request sequence*. Thus, our results extend to non-discrete and infinite metric spaces.

An obvious extension of our model is to allow the rental cost, or the cost of cloning or merging servers, to be a function of the location of the server. Certainly, this extension is needed to capture in full generality the video-on-demand application of Papadimitriou et al [24]. While some of our results do generalize to this extended model, it remains an interesting open question to solve it fully.

3 The Space-Time Product Formulation

We begin by presenting a formulation of the dynamic servers problem in terms of a product of the metric space and the time axis. This formulation is essential for describing and analyzing our algorithms in both the offline and the online setting. Suppose that the underlying metric space is the line ℓ, i.e., the 1-dimensional Euclidean space. The *space-time product* is the two-dimensional metric space $\ell \times \tau$, where $\tau = \Re^+$ and distances are defined by the L_1 metric. The coordinate axes x and y correspond to ℓ and τ, respectively. It will be convenient to assume that time flows *downwards*, i.e., in a geometric visualization the positive y axis points downwards. An event occurring at point $p \in \ell$ at time t is represented at the point $(p, t) \in \ell \times \tau$. Let ℓ_t be the horizontal line with y coordinate t. We will refer to ℓ_t as *level* t. We say that ℓ_t *lies above* $\ell_{t'}$ if $t < t'$. We plot the requests in batch t as points on the line ℓ_t. In the online setting, the requested points appear in the order of the levels $\ell_0, \ell_1, \ldots, \ell_m$.and are arbitrarily ordered within a level.

This definition generalizes to any metric space $\mathcal{M}$, albeit without the geometric intuition. We define the space-time product $\mathcal{M} \times \tau$ with points of the form (p, t), where $p \in \mathcal{M}$ and $t \in \Re^+$. The distance between (p_1, t_1) and (p_2, t_2) is defined to be $d(p_1, p_2) + |t_2 - t_1|$. We define a level $\mathcal{M}_t$ as $\mathcal{M} \times \{t\}$.

Define a *monotone path* in $\ell \times \tau$ to be a directed rectilinear path such that all vertical segments of the path are oriented in the same direction (upwards or downwards). Similarly, a monotone path in $\mathcal{M} \times \tau$ is a directed path through the requested points in $\mathcal{M} \times \tau$ which visits the requested points in order (downwards), or reverse order (upwards), of arrival time. Given a set R of requests in the product space, an *upward monotone tree* is a (directed) tree spanning R such that every point $p \in R$ has a monotone path going up to ℓ_0. Similarly, a *downward monotone tree* is a (directed) tree spanning R such that every point $p \in R$ has a monotone path going down to ℓ_m. A monotone tree (upward or downward) is said to be *minimal* if every leaf is a request in R. Observe that given any monotone tree, we can prune it to obtain a minimal tree.

The following lemma connects trees to dynamic servers.

LEMMA 3.1. *If T_1 is a minimal upward monotone tree for R and T_2 is a minimal downward monotone tree for R, then $T_1 \cup T_2$ can be interpreted as a solution for the dynamic servers problem. Furthermore, the cost of the solution is at most the sum of the lengths of the trees T_1 and T_2.*

Proof. For ease of exposition, we present the proof for the line ℓ, but it generalizes to any metric space

$\mathcal{M}$. We view a graph G spanning R in the product metric $\ell \times \tau$ as a solution for the dynamic servers problem as follows. The *horizontal* edges of G within level t represent the server movements to process batch t at time t; a horizontal edge of the form $((p,t),(q,t))$ represents the fact that *some* server moved from p to q at time t. The vertical edges of G from level t to $t+1$ represent the servers for which we pay rental cost at time t; a vertical edge $((p,t),(p,t+1))$ represents a server at point p for which we paid rental cost at time t. With this in mind, we can view the minimal upward monotone tree T_1 as specifying a method of producing a server at point p at time t for each request point (p,t).

Assume for the moment that servers can disappear into thin air. Given T_1, we can specify a method of moving and cloning servers (starting from the initial configuration) such that the following hold after the processing of the tth batch of requests,

- For each request point (p,t), we create a server at point p. For now, assume that these servers disappear immediately.
- For each edge $((q,t),(q,t+1))$, we create a server at point q. We pay rental cost for each of these servers and they survive until time $t+1$. Thus, the rental cost paid is equal to the sum of the costs of the vertical edges of T_1 from level t to $t+1$.
- For each edge $((p,t),(q,t))$, exactly one server moved from p to q while processing batch t. Hence, the movement cost is exactly the sum of the costs of the horizontal edges of T_1 within level t.

This gives a method for generating a server for each request point with cost equal to the cost of T_1. We still have to fix the assumption that servers corresponding to request points disappear into thin air.

If we reverse the time axis, by the same argument that we gave for T_1, the downward monotone tree T_2 can be viewed as creating servers in the reverse order of time, i.e., beginning from the final configuration at the last time instant and going back in time, creating a server at every request point (p,t) in reverse order of time. The cost incurred by this process is at most the cost of T_2. As in the case of T_1, the servers corresponding to requests disappear magically. Now, flip the time axis back again to its correct orientation and reverse the actions of the server cloning process corresponding to T_2. Viewed in this way, servers appear magically at each request point and T_2 specifies a method of merging these servers and ending up in the final configuration such that the total cost paid is the cost of T_2.

Now the role played by T_1 and T_2 is clear: T_1 specifies a way to create a server at every request point by performing clonings, while T_2 specifies a way to take each of these servers created by T_1 and destroy them by performing merges. Each request in R is satisfied in this manner. Hence, $T_1 \cup T_2$ is a valid solution for the dynamic servers problem and the cost of this solution is the sum of the costs of T_1 and T_2. □

4 The Offline Problem

We first consider the offline problem, giving an $O(\log n)$-approximation algorithm for general metric spaces. Later, in Section 7 we give a 6-approximation algorithm for the line. Earlier the only result known was a $O(\log n)$-approximation algorithm for the line [12]. We also show that the algorithm extends to the case where the rental cost is a function of the location of a server and thus applies to the video-on-demand application [24], giving a 3-approximation algorithm. For this problem, approximation algorithms were known only in the special case in which all requests occur at an endpoint of a path. Finally, we obtain an approximation algorithm for the problem of partitioning a rectangle into monotone rectilinear polygons [19, 7, 11]; the details are deferred to the final version of the paper.

The upward monotone trees used to construct our offline solution are similar to the well-studied rectilinear Steiner trees [5], and rectilinear Steiner arborescences (RSA) [25]. A natural question is whether existing algorithms for Steiner trees and arborescences can be modified to compute upward monotone trees. Unfortunately, this is ruled out by the following result.

THEOREM 4.1. *There exist point sets for which the cost of an optimal RSA is $\Omega(\log n)$ times the cost of an optimal upward monotone tree. Further, there exist point sets where the cost of an optimal upward monotone tree is $\Omega(\log n/\log\log n)$ times the cost of an optimal Steiner tree.*

For the Steiner tree problem on a set S, constructing an MST of the subgraph induced on S (without Steiner points) yields a 2-approximation to the minimum-cost Steiner tree for S [5]. Unfortunately, the same trick does not work for the upward monotone tree, even given Edmonds' algorithm [8] for minimum-cost arborescence when all vertices in the graph must be spanned.

THEOREM 4.2. *There exist point sets where an optimal upward monotone tree without Steiner points is $\Omega(\log n)$ times more expensive than an optimal solution with Steiner points. In fact, for the general Steiner arborescence problem, the gap can be as large as $\Omega(n)$.*

4.1 Approximation Algorithm for General Metric Spaces

We present an offline algorithm that delivers an $O(\min\{\log n, \log\rho\})$-approximation for the offline problem in arbitrary metric spaces when given a total of n requests, where ρ is the normalized diameter of the subspace presented in the request sequence. For ease of exposition, we describe our algorithm for the line ℓ; the extension to general metric spaces will be provided in the final version.

Algorithm Two-Trees works as follows. We construct tree T_1 by considering the requests in the order of the levels $\ell_1, \ldots, \ell_m$, the points in a single level being ordered arbitrarily. When point p is considered, it is greedily connected to the closest request point considered previously, by a shortest rectilinear path. Then we consider the points in the reverse order of levels $\ell_m, \ldots, \ell_1$, and construct tree T_2 greedily as before. While there is a simple proof showing that a greedy Steiner tree is an $O(\log n)$-approximation [1], we have to work harder to obtain the diameter bound and combine the two trees. (The result for the offline greedy algorithm will be important when establishing the online upper bound in Section 5; there, we will relate the cost of the online algorithm to the cost of a greedy Steiner tree.) We omit the proof of the next lemma.

LEMMA 4.1. *Tree T_1 is a minimal upward monotone tree and tree T_2 is a minimal downward monotone tree.*

Lemmas 3.1 and 4.1 show that $T_1 \cup T_2$ is a valid solution, implying the correctness of the algorithm. For the approximation, we summarize our result in the following theorem; the proof is deferred to the full version.

THEOREM 4.3. *Algorithm Two-Trees is an $O(\min\{\log n, \log \rho\})$-approximation algorithm.*

5 The Online Problem

We describe an $O(\min\{\log n, \log \rho\})$-competitive algorithm, Algorithm Counter, for the online problem in arbitrary metric spaces. For ease of exposition, we analyze a *continuous-time* version of the algorithm which can move and destroy servers at any time $t \in \Re^+$. For each server s, the algorithm pays rental cost equal to the time elapsed between the creation and the deletion of s; clearly, this need not be an integer. We will show that the continuous algorithm can be simulated by a *discrete algorithm* at no larger cost.

Algorithm Counter operates as follows. For each server s, it maintains a counter $c(s)$. Intuitively, this counter keeps track of the rental cost paid for that server thus far. The counter values increase uniformly with time, i.e., in an interval $[t, t + \mathrm{d}t]$ all counter values are increased by a value $\mathrm{d}t$. This increase in counter values continues throughout and may only be interrupted momentarily by one of the following events, resulting in a change in the server configurations. *(a). A request occurs at point p at time t.* In this case, the algorithm clones a copy of the nearest server and moves the clone to p. If multiple requests occur at time t, they may be serviced in any order. A new server is created for each request; there could be two servers at the same point just after a request has been serviced. There is a 1-to-1 correspondence between servers and requests; if server s was created to service request r, we say that r is the request *corresponding* to s. *(b). For some two servers s and s' with $c(s') \geq c(s)$, we have $c(s') \geq d(s, s')$.* In this case, Counter moves s' to s and merges the two servers, destroying s'; immediately, $c(s)$ is reset to 0. Thus, for any active server s at time t, the counter value $c(s)$ is exactly equal to the rental cost paid for s since the last merger into it.

We simulate the continuous algorithm by a discrete algorithm as follows: during time interval $(t-1, t)$, maintain the same configuration as the continuous algorithm at time t^-, i.e., just prior to servicing batch t; mimic the moves of the continuous algorithm to service the requests in batch t at time t; then, simulate the behavior of the continuous algorithm in the interval $(t, t+1)$, moving and deleting servers so that we reach the configuration of the continuous algorithm at time $(t+1)^-$. Note that this entire simulation occurs at time t itself. At this point, the algorithm pays rental cost equal to the number of active servers, i.e., the number of servers in the configuration of the continuous algorithm at time $(t+1)^-$. Thereafter, it maintains this configuration in the interval $(t, t+1)$. Since the continuous algorithm cannot increase the number of servers in the interval $(t, t+1)$, the rental cost of the discrete algorithm is at most the rental cost paid by the continuous algorithm; clearly, the server movement costs for the two algorithms are identical.

In the rest of this section, we only consider the continuous version of Algorithm Counter. We will prove that this algorithm is $O(\min\{\log n, \log \rho\})$-competitive, where n is the number of requests and ρ is the normalized diameter of the requested point set. For ease of exposition, we present the analysis only for the case of the line ℓ. In this case, we can view the problem in terms of the rectilinear Steiner tree problem in the product space. Impose a rectilinear grid on the product space by drawing vertical lines through each requested point. We view the online algorithm as constructing a rectilinear subgraph of this grid.

The proof consists of the following two parts. In Lemma 5.1, we show that the *servicing* cost, i.e., the cost of server movements during the creation of new servers, is comparable to the cost of a greedy Steiner tree for the case where the input points appear in the order of their actual arrival times. Then, in Lemma 5.2 we show that the sum of the *rental* and *merger* costs, i.e., the server movement costs during the destruction of servers, is comparable to the cost of a greedy Steiner tree for the case where the input points appear in a *slightly permuted version of the reverse order* of batches.

We can now complete the proof of the main result. Let OPT' be the cost of an optimal rectilinear Steiner tree of the n points in the request sequence, and let OPT be the cost of the optimal solution for the dynamic servers problem. We claim that $\mathrm{OPT} \geq \mathrm{OPT}'$. Based on Theorem 4.3, we further claim that the cost of a greedy Steiner tree is bounded by $O(\min\{\log n, \log \rho\})$ times the cost of the optimal Steiner tree on the requests. By Lemmas 5.1 and 5.2 proved below, this implies the following theorem.

THEOREM 5.1. *Algorithm Counter is $O(\min\{\log n, \log \rho\})$-competitive in any metric space.*

LEMMA 5.1. *The total cost of servicing the requests is at most $O(\min\{\log n, \log \rho\} \times \mathrm{OPT}')$.*

Proof. For a request $r = (p,t)$, define $\mathrm{CONE}(r)$ to be the set of points (q,t') such that $d(p,q) \leq t' - t$. Observe that if $s \in \mathrm{CONE}(r)$ then $\mathrm{CONE}(s) \subseteq \mathrm{CONE}(r)$. With every server s present at time t, we will associate a set $R_s(t)$ of requests. Server s is said to be the *representative* of every request $r \in R_s(t)$. We ensure that every request r seen before time t has some representative server at time t. Further, we will show that the representative of r is always contained in $\mathrm{CONE}(r)$. (Refer to Figure 1.)

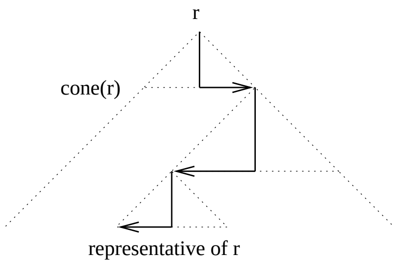

Figure 1: Request r's *representative* is in $\mathrm{CONE}(r)$.

When a server s is created to service request p at time t, set $R_s(t) = \{p\}$. The set $R_s(t)$ does not change unless a server merges with s. When a server s' merges with s, we add the requests in $R_{s'}(t)$ to $R_s(t)$. If a server s' merges with a server s at point p at time t, $(p,t) \in \mathrm{CONE}(x)$ where x was the last reset point for server s'. Consider a server s at point p at time t. We will prove that $(p,t) \in \mathrm{CONE}(x)$ for all requests $x \in R_s(t)$. The proof proceeds by induction on t. This is clearly true initially when s is created. Also, if $R_s(t)$ does not change, this is certainly true. Now if a server s' merges with s at point p, time t, then we must prove that $(p,t) \in \mathrm{CONE}(x)$ for all requests $x \in R_{s'}(t)$. Let y at time t' be the last reset point of s'; clearly, $R_{s'}(t') = R_{s'}(t)$. Consider any request $x \in R_{s'}(t)$. We have that $y \in \mathrm{CONE}(x)$ and $(p,t) \in \mathrm{CONE}(y)$. This implies that $(p,t) \in \mathrm{CONE}(x)$.

Consider the following greedy Steiner tree algorithm: examining the requests in the same order as the online algorithm, construct a spanning tree on the requests by connecting each request p to the closest request occurring before p. By Theorem 4.3, this algorithm produces a tree of cost at most $O(\min\{\log n, \log \rho\})$ times the cost of an optimal Steiner tree on the requested points.

Consider a request $r = (p,t)$. Let $r' = (q,t')$ be the request nearest to r that arrives before it. Let the representative of r' be at point x at time t. Then, $(x,t) \in \mathrm{CONE}(r')$, which implies that $d(q,x) \leq t - t'$. The cost for the greedy Steiner tree algorithm to connect r to the Steiner tree is $C_{\mathrm{greedy}}(r) = d(p,q) + t - t'$. On the other hand, the cost C_{service} paid by the online server algorithm to service r is the distance from p to the nearest server present at time t. Since a server is present at point x at time t, we have that

$$\begin{aligned} C_{\mathrm{service}}(r) \leq d(p,x) &\leq d(p,q) + d(q,x) \\ &\leq d(p,q) + t - t' = C_{\mathrm{greedy}}(r). \end{aligned}$$

The total service cost is at most the cost of a greedy Steiner tree, and the result follows by Theorem 4.3. □

LEMMA 5.2. *The sum of the total rental and merger costs is bounded by $O(\min\{\log n, \log \rho\} \times \mathrm{OPT}')$.*

Proof. Consider a server s' which merges with a server s at time t. Let C and C' be the rental costs paid for s and s', respectively, since their last reset points. We claim that $C \leq C'$ and that the distance moved to merge the two servers is at most C'. We *assign* cost C' to server s'; call this the *assigned cost* for s' and is denoted by $C_a(s')$. Observe that $C_a(s')$ is simply $c(s')$ when s' merges into another server s. We claim that 3 times the assigned cost is enough to pay for the rental costs of the two servers s and s' since their last reset times, as well as the cost of merging s' with s. This justifies resetting the counter for s to 0.

This scheme ensures that for each server, the total rental and merging cost is accounted for as follows (refer to Figure 2). For a server s, let t_0 be its creation time, $t_1, \ldots, t_{q-1}$ be the various reset times (caused by other servers merging into s), and t_q be the time at which s itself merges into some other server. Then, the rental cost paid for s during the time interval $[t_{i-1}, t_i)$, for $1 \leq i \leq q-1$, is assigned to the server which merges into s at time t_i. The rental cost for s during time interval $[t_{q-1}, t_q)$ is assigned to s itself, as also the cost of the merger and the rental cost of the server that it merges into.

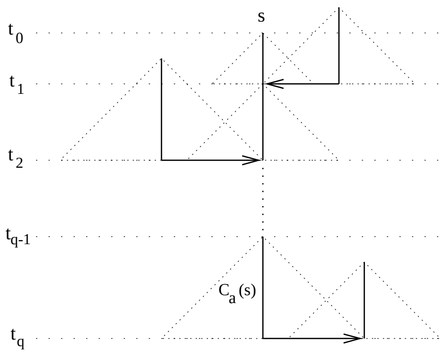

Figure 2: Rental/merger cost accounting for server s.

Consider the requests in reverse order of the merging times of the servers corresponding to them. We will prove that the greedy Steiner tree algorithm, when given the requested points in this order, pays a cost at least $C_a(s)$ for the request r corresponding to s.

Consider a server s which merges into another server at time t, where s is located at a point p just prior to the merger. Let r be the request corresponding to server s. When r is seen by the greedy algorithm, the request points previously seen by the greedy Steiner tree algorithm are those whose corresponding servers are present at time t, as well as all request points that appear *after* time t. Let $r' = (p, t')$ be the last reset point of s before it merges and is destroyed. Consider CONE(r') truncated to time t. Now, all requests occurring after time t are outside TRUNCATED-CONE(r'). Thus, they are at a distance at least $t - t'$ from r', and hence from r. Further, all servers present at time t are also outside TRUNCATED-CONE(r'). Hence, the corresponding requests occur at points whose distance is at least $t - t'$ from s in the metric space; otherwise, the two servers would have been merged by the algorithm. Therefore, the request closest to r (amongst the requests seen before r) is at distance at least $t - t'$ from r. The greedy Steiner tree algorithm must pay at least $t - t'$ to connect r to the tree being constructed by it. Note that $t - t'$ is simply the rental cost since the last reset time for s; further, this is the assigned cost for s according to our charging scheme.

This proves that for all servers which are destroyed, Algorithm Counter's rental and merging costs are bounded by 3 times the cost incurred by the greedy Steiner tree algorithm, which in turn is $O(\log n)$ times the cost of the optimal Steiner tree on the requested points. Further, for servers s and s' which survive, $c(s) < d(s, s')$. This implies that the sum of the counter values for all surviving servers (which is the total rental cost paid for the surviving servers) is at most twice the cost of the minimum spanning tree of the positions of these servers. This in turn is a lower bound on the optimal cost, as each server is at the position of some request and the initial configuration consisted of only one server. Hence, the total rental cost paid for the surviving servers only leads to an additive term in the competitive ratio.

We still need to establish that the total rental and merging cost is also within a factor $O(\log \rho)$ of the cost of the optimal Steiner tree. To this end, construct a subset R' of the set R of requests as follows: consider the requests in the reverse order of arrival, i.e., in order $\ell_m, \ldots \ell_0$; initially R' is empty; when a request in R is considered, place it in R' if there is no request currently in R' with rectilinear distance strictly less than δ from it. Define $n' = |R'|$. By construction, the minimum distance between any two requests in R' is at least δ. The optimal offline algorithm must pay at least $\delta/2$ for each request in R'; hence OPT is at least $n'\delta/2$.

We will bound $\sum C_a(s)$ separately for servers corresponding to requests in R' and in $R \setminus R'$. We first claim that for every request $r_1 \notin R'$, there must be a *unique* request $r_0 \in R$ below r_1 and within distance less than δ of it. Clearly, r_0 cannot be on the same level as r_1 since δ is the minimum inter-point distance; for the same reason, the only point at any level below that is within distance δ of r_1 must project up to r_1 itself. That is, both r_0 and r_1 must occur at the same point on ℓ, say p. If $r_0 = (p, t_0)$ and $r_1 = (p, t_1)$ where $t_0 > t_1$, then $t_0 - t_1 \leq \delta$. We will say that r_1 is a *dependent* of r_0. We charge the cost assigned to the server corresponding to r_1 to the request r_0.

We now show that the total cost charged to any request in R' is at most δ. Consider a request $r_0 = (p, t_0) \in R'$. Its dependents must be of the form $(p, t_q), \ldots, (p, t_1)$, where $t_0 - \delta \leq t_q \leq \ldots \leq t_1 \leq t_0$. The server corresponding to (p, t_i) must merge with the server corresponding to (p, t_{i-1}) at time t_{i-1}. Note that the cost assigned to the server corresponding to (p, t_i), for $1 \leq i \leq q$, is at most $t_{i-1} - t_i$. Thus, the total cost charged to any request $r \in R'$ is at most $\sum_{i=1}^{q}(t_{i-1} - t_i) \leq t_0 - t_q \leq \delta$. We can bound the total cost $\sum C_a(s)$ assigned to servers s corresponding to requests in $R \setminus R'$ by $n'\delta$, which is at most $2 \times$ OPT.

We now bound $\sum C_a(s)$ for servers s corresponding to requests in R'. Assume, without loss of generality, that there is no time period of length Δ where no requests arrive. We claim that the assigned cost $C_a(s)$ for a server s never exceeds Δ. By the previous argument, we have that $C_a(s) \leq C_{\text{greedy}}(s)$. It follows that $C_a(s) \leq \min\{C_{\text{greedy}}(s), \Delta\}$. Therefore, $C' = \sum \min\{C_{\text{greedy}}(s), \Delta\}$ is an upper bound for $\sum C_a(s)$. It can be shown, using an analysis very similar to that for Theorem 4.3, that C' is bounded by $O(\log \rho)$ times the cost of the optimal Steiner tree on the requests. We omit the details. $\square$

6 Lower Bound for Online Algorithms

The results of Berman and Coulston [4] imply a lower bound of $\sqrt{\log \rho}$ for dynamic servers on the line without deletion costs. However their arguments do not extend to the case when deletions are not free. We outline a considerably more complicated construction of a lower bound of $\frac{\log\log\rho}{\log\log\log\rho}$ for dynamic servers with deletion costs. Detailed proofs will appear in the full version.

Given an online algorithm, we describe an adversary which produces a request sequence. We restrict our attention to the metric space consisting of a discrete line segment of length $\Delta = 2^r$. In this case, $\delta = 1$, so $\rho = \Delta$. Let P_i be the set of $2^i + 1$ points that partition the line into 2^i intervals of length 2^{r-i}; note that $P_1 \subset P_2 \subset \ldots \subset P_r$. The requests produced in any batch consist of the points P_i for some i. The adversary constructs the request sequence as a sequence of phases. A single phase is a sequence of batches of requests of the form $P_{i_1}, P_{i_2}, \ldots, P_{i_m}$, where $i_1 \le i_2 \le \ldots \le i_m$.

We describe a strategy to force any deterministic algorithm to pay a cost $\Omega(k\rho)$ per phase; we specify k later. We ensure that the cost incurred by the adversary per phase is at most $O(\rho)$. This implies a lower bound of $\Omega(k)$ on the competitive ratio. The idea is to produce a request sequence which forces the algorithm to either make k configuration changes, each costing $\Omega(\rho)$, or pay a combined rental and deletion cost $\Omega(k\rho)$.

Consider the number of servers maintained by the algorithm as a function of time. Time t refers to the tth batch in the phase. A *threshold* is a function $F(t)$ of time of the form $F(t) = R/t$. For a value k to be specified later, we will define k thresholds $F_i(t) = R_i/t$, for $1 \le i \le k$, by choosing R_i such that $1 \le R_k \le \ldots \le R_2 \le R_1 = O(\rho)$.

As servers cannot be deleted freely, the algorithm cannot decrease the number of servers continuously without incurring a high deletion cost. To account for the deletion cost paid by the algorithm, we introduce the notion of *pseudo server count.* Partition the line into n equal intervals. The *n-occupancy fraction* is the fraction of the intervals containing a server of the algorithm. Let n_1 be the largest power of 2 for which the n_1-occupancy fraction is $\ge 8/9$. The pseudo server count is said to be n_1, until the n_1-occupancy fraction falls below $1/9$. Then, let n_2 be the largest power of 2 for which the n_2-occupancy fraction is $\ge 8/9$. Then, the pseudo server count is said to be n_2, until the n_2 occupancy fraction falls below $1/9$, and so on.

LEMMA 6.1. *If the n-occupancy fraction falls from $\ge 8/9$ to $\le 1/9$, the algorithm incurs cost at least $\rho/9$.*

If the pseudo server count is below $F_i(t)/4$ at time t, we say that the online algorithm is below the ith threshold at time t. Also, if the number of requests issued by the adversary in a batch t is greater (lesser) than $F_i(t)/4$, we say that the adversary is above (below) the ith threshold. Define $P(x)$ to be the largest P_i whose size does not exceed x; let $n(x) = |P(x)|$. Note that $\frac{x}{2} \le n(x) \le x$. The adversary is said to move to the ith threshold at time t, if it places requests at the points in $P(F_i(t))$ at time t. The requests at time $t+1$ are at the same points as the requests at time t unless the adversary moves to a new threshold at time $t+1$. We say that the adversary is at the ith threshold at time t if the adversary moved to the ith threshold at some time $t' \le t$ and did not move to the next threshold between t' and t.

The adversary strategy is as follows. At time 1, the adversary is at the kth threshold. In general, if the adversary is at the ith threshold, it waits for the first time instant when either the algorithm falls below the $(i-1)$st threshold, or the adversary is above the $(i-1)$st threshold. If the algorithm falls below the ith threshold at time t, the adversary moves to the $(i-1)$st threshold at time $t+1$. On the other hand, if the adversary goes above the $(i-1)$st threshold first, the adversary ends the phase and begins a new one. Also, if the adversary ever reaches the first threshold, it ends the phase and begins a new one. Each time the adversary moves to a new threshold or begins a new phase, the pseudo server count is recomputed.

LEMMA 6.2. *The optimal offline cost per phase is at most $O(\rho)$.*

Let $y_i = 8\rho/R_i$; set $y_1 = 2^k$, $y_i = (y_{i-1})^k$; hence, $y_i = 2^{k^i}$. We choose the maximum possible value for k such that $y_k \le \rho$; thus, $k = \Omega(\frac{\log\log\rho}{\log\log\log\rho})$. Note that our choice of y_i values ensures that the corresponding R_i values satisfy the condition $1 \le R_k \le \cdots \le R_2 \le R_1 = O(\rho)$ as required.

LEMMA 6.3. *When the adversary moves from threshold i to $i-1$, the algorithm pays service cost $\Omega(\rho)$.*

LEMMA 6.4. *If the adversary is at a particular threshold i and the algorithm never falls below the threshold $i-1$, the algorithm pays a combined rental and deletion cost $\Omega(k\rho)$.*

LEMMA 6.5. *The online algorithm pays cost $\Omega(k\rho)$ per phase.*

THEOREM 6.1. *There is a lower bound of $\Omega(k) = \Omega(\frac{\log\log\rho}{\log\log\log\rho})$ for any online dynamic servers algorithm.*

7 Improved Approximation for the Line

We now present a 6-approximation algorithm in the case of the line. The improvement is based on a 3-approximation algorithm, called Algorithm Triangle, for the offline, upward monotone Steiner tree problem.

For a point p, consider the points above p (including those at the same horizontal level as p) and at a rectilinear distance at most d from it. These points form an isosceles triangle with p as the midpoint of its horizontal base. This triangle of size d about p is referred to as $T(p,d)$. Note that p is at a rectilinear distance d from the boundary of $T(p,d)$. (Refer to Figure 3.)

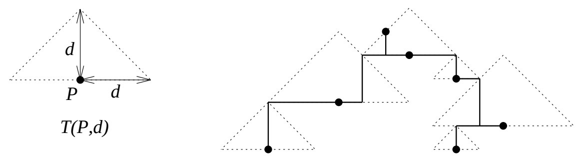

Figure 3: *The Triangle Construction.*

Algorithm Triangle works as follows. Consider the points in a top-down order, arbitrarily ordering the points within a level. The algorithm progressively produces an upward monotone tree spanning all points seen so far. Further for every point p, it constructs a triangle $T(p,d_p)$ where the parameters d_p will be specified later. The interiors of the triangles are disjoint. For every point p, the base of $T(p,d_p)$ is a part of the tree constructed up to that point. When considering a new point p, it *grows* a triangle about p, i.e., increases d till $T(p,d)$ intersects a previously constructed triangle T'. The minimum value of d required for such an intersection is defined to be d_p. Observe that the intersection point must lie on the base of T', already a part of the tree. Connect p to this intersection point by a monotone path of length d_p. Also add the base of $T(p,d_p)$ to the tree; the base is of length $2d_p$. The total cost incurred for p is at most $3d_p$. Due to multiple points on the same level, a new point q may already lie on the base of an existing triangle; in that case, $d_p = 0$ and $T(p,d_p)$ is degenerate. We claim that this algorithm produces a valid solution.

For the approximation bound, consider the optimal tree T. For each point p, consider the monotone path in T connecting it to level ℓ_0. This path must leave $T(p,d_p)$ at some point above p. The portion of this path within $T(p,d_p)$ must have length at least d_p; charge this cost to triangle $T(p,d_p)$. By construction, the triangles are disjoint and therefore no portion of the optimal tree is charged to more than one triangle. Thus, the sum of the sizes of the triangles is a lower bound on OPT. The algorithm pays $3d_p$ for each triangle of size d_p, giving a 3-approximation.

An obvious optimization would be to prune the tree constructed by Algorithm Triangle to obtain a minimal tree. This could be done by deleting unnecessary portions of the base of each triangle $T(p,d_p)$, as shown in Figure 3. We can show a lower bound of 2.5 on the approximation ratio of the optimized Algorithm Triangle. We use the optimized Algorithm Triangle to construct minimal upward and downward monotone trees on the request points and take their union. Lemma 3.1 implies that this gives a 6-approximation algorithm for dynamic servers on the line.

THEOREM 7.1. *Algorithm Triangle gives a 3-approximation for the upward monotone Steiner tree problem, and 6-approximation to offline dynamic servers on the line..*

7.1 Non-uniform Rental Costs and Video-on-Demand

We can apply Algorithm Triangle to the case where different points have different rental costs, when the metric space is the line. This corresponds to the video-on-demand problem considered by Papadimitriou et al [24]; in this application, there is no deletion cost. The equivalent problem in the space-time product space is to construct an upward monotone Steiner tree on the requests. As before, we define the triangle $T(p,d)$ to be the set of points which are *above* p and at distance at most d from it, where the distances are with respect to shortest paths in the space-time product graph. The distance metric in the space-time product graph is now more complicated and must take into account the differing rental costs:

$$d((p,t),(q,t')) = \min_x [d(p,x) + d(x,q) + \mathrm{rent}(x)(t'-t)]$$

Due to this, the triangles $T(p,d)$ are not literally triangles but have more complex shapes. These shapes have the property that the base is a straight line and the upper boundary a polygon. The generalized triangles satisfy the additional property that for any point q on the upper boundary, the perpendicular from p to the base is completely contained in $T(p,d)$. The algorithm proceeds by growing triangles as before. By the property of generalized triangles, a *growing* triangle will always intersect a previously-constructed triangle at its base. Everything else goes through as before, and the algorithm gives a 3-approximation. We claim that the analysis holds even when rental costs depend on time.

COROLLARY 7.1. *Algorithm Triangle gives a 3-approximation even when the rental costs are a function of server locations and of time.*

Acknowledgements

We thank Chandra Chekuri and Sudipto Guha for helpful discussions.

References

[1] N. Alon and Y. Azar. On-line Steiner trees in the Euclidean plane. *Discrete & Computational Geometry*, 10 (1993), pp. 113–121.

[2] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 1993, pp. 164–173.

[3] Y. Bartal, A. Fiat, and Y. Rabani. Competitive Algorithms for Distributed Data Management. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, 1992, pp. 39–50.

[4] P. Berman and C. Coulston. On-line Algorithms for Steiner Tree Problems. To appear in *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.

[5] M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In: D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[6] G.S. Chirikjian and J.W. Burdick. Kinematics of Hyper-Redundant Manipulators. In *Proceedings of the 2nd International Workshop on Advances in Robot Kinematics*, 1990, pp. 392–399.

[7] D.-Z. Du, L.-Q. Pan, and M.-T. Shing. Minimum Edge Length Guillotine Rectangular Partition. Technical Report MSRI 02418-86, MSRI, Berkeley, 1986.

[8] J. Edmonds. Optimum Branchings. *Journal of Research of the National Bureau of Standards B*, 71B (1967), pp. 233–240.

[9] P. Finn, D. Halperin, L. Kavraki, J-C. Latombe, R. Motwani, C. Shelton, and S. Venkatasubramanian. Geometric Manipulation of Flexible Molecules. In *Proceedings of the ACM Workshop on Applied Computational Geometry*, 1996.

[10] T. Fukuda and S. Nakagawa. Dynamically Reconfigurable Robotic Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1988, pp. 1581–1586.

[11] T. Gonzalez, M. Razzazi, and S.-I. Zheng. An Efficient Divide-and-Conquer Approximation for Hyperrectangular Partitions. In *Proceedings of the 2nd Canadian Conference on Computational Geometry*, 1990, pp. 214–217.

[12] D. Halperin, J.-C. Latombe, and R. Motwani. Dynamic Maintenance of Kinematic Structures. In: J.P. Laumond and M. Overmars, editors, *Algorithmic Foundations of Robotics.* A.K. Peters Publishing, 1996. Preliminary version in *Proceedings of the 2nd International Workshop on the Algorithmic Foundations of Robotics*, Toulouse, France, 1996.

[13] D. Halperin and M.H. Overmars. Spheres, Molecules, and Hidden Surface Removal. In *Proceedings of the 10th ACM Symposium on Computational Geometry*, 1994, pp. 113–122.

[14] F.K. Hwang. An $O(n \log n)$ Algorithm for Rectilinear Minimal Spanning Tree. *Journal of the ACM*, 26 (1979), pp. 177–182.

[15] S. Irani and A. Karlin. Online Computation. In: D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1996.

[16] P. Finn, L.E. Kavraki, J-C. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian and A. Yao. RAPID: Randomized Pharmacophore Identification in Drug Design. to appear in *Proceedings of 13th Annual ACM Symposium on Computational Geometry*, 1997.

[17] Y. Koga, K. Kondo, J. Kuffner, and J-C. Latombe. Planning Motions with Intentions. In *Proceedings of SIGGRAPH'94*, 1994, pp 395–408.

[18] E. Koutsoupias and C.H. Papadimitriou. On the k-server conjecture. In *Proceedings of 26th Annual ACM Symposium on Theory of Computing*, 1994, pp. 507–511.

[19] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir. Minimum Edge Length Partitioning of Rectilinear Polygons. In *Proceedings of the 20th Annual Allerton Conference on Communication, Control, and Computing*, 1985, pp. 53–63.

[20] S. Loeb. Architecting Personalized Delivery of Multimedia Information. *Communications of the ACM*, 35 (1992), pp. 39–48.

[21] M. Manasse, L. McGeoch, and D.D. Sleator. Competitive Algorithms for Server Problems. *Journal of Algorithms*, 11 (1990), pp. 208–230.

[22] C.H. Papadimitriou, S. Ramanathan, and P.V. Rangan. Information Caching for Delivery of Personalized Video Programs for Home Entertainment Channels. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, 1994, pp. 214–223.

[23] C.H. Papadimitriou, S. Ramanathan, P.V. Rangan, and S. Sampathkumar. Multimedia Information Caching for Personalized Video-on-Demand. *Computer Communications*, 18 (1995), pp. 204–216.

[24] C.H. Papadimitriou, S. Ramanathan, and P.V. Rangan. Optimal Information Delivery. In *Proceedings of the 6th Annual International Symposium on Algorithms and Computation*, 1995, pp. 181–187.

[25] S.K. Rao, P. Sadayappan, F.K. Hwang, and P.W. Shor. The Rectilinear Steiner Arborescence Problem. *Algorithmica*, 7 (1992), pp. 277–288.

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

[27] V. Vassalos. Personal Communication, 1997.

[28] M. Yim. Locomotion with a Unit-Modular Reconfigurable Robot. PhD thesis, Stanford Technical Report STAN-CS-94-1536, Department of Computer Science, Stanford University, 1994.