2008

The Online Transportation Problem: On the Exponential Boost of One Extra Server

Christine Chung
Connecticut College, cchung@conncoll.edu

Patchrawat Uthaisombut

Kirk Pruhs

The Online Transportation Problem: On the Exponential Boost of One Extra Server

Keywords
online transportation

Comments
Presented at LATIN 2008 (Latin American Symposium on Theoretical Informatics).

The Online Transportation Problem: On the Exponential Boost of One Extra Server *

Christine Chung *Kirk Pruhs* *Patchrawat Uthaisombut*

Abstract

We present a poly-log-competitive deterministic online algorithm for the online transportation problem on hierarchically separated trees when the online algorithm has one extra server per site. Using metric embedding results in the literature, one can then obtain a poly-log-competitive randomized online algorithm for the online transportation on an arbitrary metric space when the online algorithm has one extra server per site.

1 Introduction

The setting for the online transportation problem is a collection of k server sites located in some metric space. Points in the metric space, which represent requests for service, arrive online over time. Server site j has a fixed capacity B_j specifying the number of requests that can be handled by site j. After each request arrives, the online algorithm must irrevocably match that request to a single server site that has not yet reached its capacity. Conceptually it is convenient to think of B_j as the number of servers at site j, and one of these servers traveling to a request that is assigned to site j. The objective is to minimize the total (or equivalently average) distance between the requests and their assigned server sites.

If the locations of the requests are known a priori, then this is the standard offline transportation problem [9, 11]. Many of the applications of the transportation problem are naturally online problems. We give two examples here. In the first example, the server sites could be fire stations, the capacity of a fire station could be the number of fire trucks stationed there, and the requests could be the location of a fire. The objective would then be to minimize the average distance that fire trucks travel. In the second example, the server sites could be schools, the capacity of a school could be the number of students that the school can handle, and the requests could be the locations of new students that move to the school district. The objective would then be to minimize the average distance between students and their assigned schools.

1.1 Previous Results

The online weighted matching problem is a special case of the online transportation problem in which each server site has unit capacity. The competitive ratio of every deterministic algorithm for online matching, and hence for online transportation, is at least $2k - 1$ [6, 10]. The metric space in the lower bound instances that establish this bound is a star. In a star there is a unique root point that is a unit distance from all other points, and all pairs of non-center points, called leaves, are distance two from each other. [6, 10] give a $(2k - 1)$-competitive algorithm, called Permutation in [6], for online matching. However, the competitive ratio of Permutation for the more general online transportation problem is $\Theta(B)$, where $B = \sum_{i=1}^{k} B_i$ is the aggregate capacities of the server sites [7]. In contrast, [6] shows that the competitive ratio of the natural

*Computer Science Department. University of Pittsburgh. {chung,kirk,utp}@cs.pitt.edu. Supported in part by NSF grants CNS-0325353, CCF-0448196, CCF-0514058 and IIS-0534531.

greedy algorithm, that always matches each request to an arbitrary nearest server site with residual capacity, is $2^k - 1$ for online transportation.

The fact that the optimal deterministic competitive ratios are so high prompted [8] to consider resource augmentation analysis for online transportation. Resource augmentation analysis compares the performance of an online algorithm to the optimal solution with less resources. In the context of online transportation, the resource is the number of servers per site. So an s-server c-competitive algorithm A guarantees that if A has $s \cdot B_j$ servers at each site j then the cost of A's assignment is at most c times the cost of the optimal assignment assuming B_j servers at each site j. [6] show that the greedy algorithm is 2-server $\Theta(\min(k, \log B))$-competitive for online transportation. [8] then considers a slightly modified greedy algorithm, that in the cases of approximate ties for the nearest server site, picks a server site that has been assigned the least requests so far. [8] show that this modified greedy algorithm is 2-server $O(1)$-competitive for online transportation. One can also see from the analyses in [8] that the greedy algorithm is 3-server $O(1)$-competitive for online transportation.

[3, 12] consider randomized algorithms for the online matching problem against an oblivious adversary that must specify the input a priori. In particular, [3, 12] consider the simple randomized greedy algorithm that services each request with an unused server site picked uniformly at random from the closest server sites. It is easy to see that the randomized greedy algorithm is $O(\log k)$-competitive for online matching in a star. [3, 12] extend this analysis to show that randomized greedy is $O(\log k)$-competitive in $(\log k)$-Hierarchically Separated Trees ($\log k$-HST's). [1, 4] show that for every metric space, there exists a probability distribution over $\log k$-HST's, for which the expected distance between any pair of points in a randomly drawn $\log k$-HST is at most $O(\log^2 k)$ times the distance between this pair of points in the metric space. Combining these two results gives an $O(\log^3 k)$-competitive randomized algorithm (note that this algorithm is not randomized greedy) against an oblivious adversary for online matching on an arbitrary metric space.

For a summary of results for online matching problems, and related online network optimization problems, see [7].

1.2 Our Results

One motivation for considering resource augmentation analysis is that it allows one to show that an algorithm is competitive, without additional resources, for instances where the optimal solution is not so sensitive to changes in the number of available servers per site. More precisely, an s-server c-competitive algorithm would be $c \cdot d$-competitive on instances where a factor of s change in the number of servers per site does not change the optimal cost by more than a factor of d. The smallest resource augmentation considered in [8] was doubling the number of servers per site. Our main aim here is consider the effect of more modest resource augmentation. More precisely, we consider the effect of adding just one additional server per site. We will say that an online algorithm A is $+1$-server c-competitive if A guarantees that if A has $B_j + 1$ servers at each site j then the cost of A's assignment is at most c times the cost of the optimal assignment assuming B_j servers at each site j. Then a $+1$-server c-competitive algorithm would be $c \cdot d$-competitive on instances where adding one server per site does not change the optimal cost by more than a factor of d.

In this paper we consider a natural deterministic online algorithm that we call Balancing of Displaced Servers (BODS). Intuitively, BODS prefers using server sites that have serviced less requests (of positive cost), as did the modified greedy algorithm in [8], but for BODS this preference is only relevant in the case of exact ties.

BODS Description: BODS always assigns the request to a nearest server with residual capacity. BODS breaks ties among closest servers by assigning the request to a server site that has been to date assigned the least number of requests with positive cost.[1] If a tie remains, BODS uses the first server site with residual capacity in some arbitrary ordering of the server sites.

[1] Note that this is not the same as assigning the request to the site that has serviced the least requests so far, since a request on site j costs nothing if assigned to site j.

From what is known to date about online transportation and matching, the hardest metric space for the online algorithm is always the star. The results in [3, 12] can be viewed as a reduction from a general metric space to the star via HST's. So we begin by first considering the star metric space. We show BODS is $+1$-server $O(\log k)$-competitive for online transportation in a star. We found it surprising that such modest resource augmentation drops the competitive ratio so dramatically, from linear in k to logarithmic in k. We then show that BODS is essentially optimally competitive. That is, we show that there is no $+1$-server $o(\log k)$-competitive deterministic online algorithm for online transportation on a star. We then generalize our analysis to show that BODS is $+1$-server $O(\log k)$-competitive for online transportation on a $\log k$-HST. Our proof proceeds by induction on the height of the HST, following the same general structure as the proof in [12]. However the introduction of resource augmentation adds some technical difficulties. For example, when the analysis in [12] considers the subinstances on the subtrees of the root, we now have to be concerned about the possibility that in some of these subinstances there are more requests than servers available to the adversary, and thus there is no feasible optimal solution without resource augmentation. Using the results in [1, 4], we obtain a $+1$-server $O(\log^3 k)$-competitive randomized algorithm for online transportation for an arbitrary metric space against an oblivious adversary.

Our results extend those in [3, 12] in two ways: (1) For arbitrary metric spaces, our analysis holds for the more general online transportation, not just for the special case of online matching, and (2) for stars and HST's, we obtain a deterministic bound on the competitive ratio instead of a randomized bound against an oblivious adversary. Of course these extensions come at the cost of requiring modest resource augmentation. Admittedly these extensions will not generally be viewed as great contributions because (1) for general metric spaces, it is quite plausible the analysis in [3, 12] could extend (with some more involved analysis) to online transportation, and (2) HST's are rather specialized metric spaces. It is common that the most lasting contribution of many papers is the interesting questions that naturally arise from the paper, instead of the actual results in the paper. In our humble opinion, we believe that this is likely the case for this paper. The interesting question that naturally arises here is:

> *Is there a $+1$-server poly-log-competitive deterministic algorithm for online transportation on an arbitrary metric space?*

Intuitively the star is the hardest metric space, and the deterministic $+1$-server $O(\log k)$-competitiveness result for a star should extend to an arbitrary metric space. But current metric embedding techniques do not seem sufficient to address this question. One can obtain a poly-log-competitive deterministic offline algorithm by deterministically generating a collection of HST's, and then using the tree that gives the best results [2]. But it is not clear how an online algorithm should learn or construct the right metric embedding online as it sees requests. So it seems like the above question could well be a vehicle to extend the current understanding of metric embeddings.[2] Alternatively, if it somehow turned out that there is no $+1$-server poly-log-competitive deterministic algorithm for online transportation on an arbitrary metric space, then this would be interesting because it would be the first example of an online matching/transportation problem where the hardest metric space was not a star.

2 Online Transportation on a Star

In this section, we assume that the k server sites are at the leaves of a star. We show that BODS is $+1$-server $O(\log k)$-competitive on a star, and that this is essentially the best possible result that one can obtain. We start with the lower bound.

[2] We are aware of the result in [5] that gives a deterministic online algorithm that uses a collection of HST's and is poly-log-competitive. In the algorithm in [5], $\log k$ different HST's are generated a priori, and then the online algorithm always uses an HST that is gauranteed to approximate the distance between points that arrive online. This does not work for online matching and online transportation because it essentially simulates the greedy algorithm, which is not competitive in a general metric space.

When a request arrives at a leaf, and a server from the leaf site is assigned to the request, we refer to it as a *local service* or *local assignment*. If a request arrives at a leaf site and a server from a different leaf site is assigned to the request, we call it a *remote service* or *remote assignment*. Thus, all root requests cost 1 to service, a leaf request serviced *locally* costs 0, and a leaf request serviced *remotely* costs 2. A request is called an *excess request* if it arrives at a server site s after B_s requests have already arrived at site s. Hence, the only requests that optimal will pay for are those that arrive at the root and those that are excess requests. We define $C_A(I)$ to be the cost of algorithm A on input I.

2.1 The General Lower Bound

Theorem 1 *There is no deterministic $+1$-server $o(\log k)$-competitive for online transportation on a uniform star with k leaves.*

Proof: Consider the input instance where for all sites j, for $1 \leq j \leq k$, we have $B_j = b$. Assume that the online algorithm has e extra servers per site (so we will eventually use $e = 1$ to prove the statement of the theorem). Let x be an integer value to be set later. First, xb requests arrive at the root node, then continue to arrive b at a time at the next leaf with the fewest remaining available servers until a total of $B = bk$ requests have been made. Call each set of b requests made from a leaf node a *hit*.

First note that the cost of the optimal offline solution C_{OPT} is always xb since it knows the request sequence in advance and for the requests from each hit it will reserve local servers from that site. So it services the xb root requests using the remaining servers that are not already reserved. Then it will be able to service requests from each hit locally. The adversary's strategy finds a value for x that maximizes the competitive ratio.

Since xb requests are initially made from the root node, and there are bk total requests, there are $k - x$ total hits. Let $f(i)$ for all $1 \leq i \leq k - x$ be the maximum (over all un-hit server sites) after hit i of the number of servers that have been used at any site by the online algorithm. Define $f(0)$ to be the maximum number of servers used at any site by the online algorithm for the initial xb root requests. Define $c(i)$ as the number of requests in hit i that are serviced remotely (at a cost of 2) by the online algorithm. By definition of $f(i)$ and the adversarial strategy outlined above, we know that $c(i) = f(i-1) - e$ for $1 \leq i \leq k - x$, where we subtract e from $f(i-1)$ because there are $b + e$ servers at each of the server sites allowing us to service e extra requests locally.

After hit i, for $1 \leq i \leq k - x$, $xb + ib$ total requests have been made, whereas at most $i(b+e)$ have been used at the i sites that have been hit so far. That leaves at least $(x+i)b - i(b+e)$ servers missing at the remaining $k - i$ unhit sites. So, by the pigeon-hole principle, for $1 \leq i \leq k - x$,

$$f(i) \geq \frac{(x+i)b - i(b+e)}{k-i} = \frac{xb - ie}{k-i}.$$

By the definition of $c(i)$, the total cost C_{ON} of the solution returned by any online algorithm ON is therefore:

$$\begin{aligned} C_{ON} &= xb + 2\sum_{i=1}^{k-x} c(i) \;=\; xb + 2\sum_{i=1}^{k-x}(f(i-1) - e) \\ &\geq xb + 2\sum_{i=0}^{k-x-1}\left(\frac{xb - ie}{k-i} - e\right) \;=\; xb + 2(xb - ek)(H_k - H_x). \\ \text{Thus,}\quad \frac{C_{ON}}{C_{OPT}} &\geq \frac{xb + 2(xb - ek)(H_k - H_x)}{xb} \;\geq\; 1 + 2\left(1 - \frac{ek}{xb}\right)\left(\ln\frac{k+1}{x} - 1\right). \end{aligned}$$

By choosing $x = \max\left\{1, \frac{2ek}{b}\right\}$, we get the following two cases.

Case 1: $b > 2ek$. Then $x = 1$ and $\frac{C_{ON}}{C_{OPT}} \geq 1 + 2\left(1 - \frac{ek}{b}\right)(\ln(k+1) - 1) \geq \ln k$.

Case 2: $b \leq 2ek$. Then $x = \frac{2ek}{b}$ and $\frac{C_{ON}}{C_{OPT}} \geq 1 + 2\left(1 - \frac{1}{2}\right)\left(\ln\frac{(k+1)b}{2ek} - 1\right) \geq \ln\frac{b}{2e}$.

Putting both cases together: $\frac{C_{ON}}{C_{OPT}} \geq \min\left(\ln k, \ln\frac{b}{2e}\right)$.

■

2.2 BODS on the Star

A *live* site is a site with at least one server still unassigned. A *dead* site is a site whose servers have all been assigned, so there are no more available at that site to service any future requests. A server is called *displaced* if it has been assigned to a remote request. We define a *restricted instance* to be one where: (A) no more than B_j requests arrive at server site j, (B) no request is serviced locally by BODS, and (C) B requests arrive. We now show, without loss of generality, that we may restrict our analysis of BODS to restricted instances.

Lemma 2 *If there is an instance I for which the $+1$-server competitive ratio of BODS is at least c, then there is a restricted instance I' for which the $+1$-server competitive ratio of BODS is at least c.*

Proof: We consider the restrictions in order. First consider restriction (A). Consider request number $B_j + i$ to site j. If $i > 1$ then this request in I' will appear at the root instead of at j. This modification will not change how BODS services requests. This will lower the cost of BODS and optimal by 1 each, which will not decrease the competitive ratio. If $i = 1$ and $B_j > 0$ then we decrement the value of B_j in I' by 1, and there is no corresponding request in I'. This modification will not change how BODS services other requests, and does not change the cost to BODS. To see that the cost of the optimal solution does not increase, let $k \neq j$ be a site that was assigned one of the $B_j + 1$ requests at B_j, and r be the request serviced by the deleted server at site j, in the optimal solution. The request r can now be assigned to site k with no increase in cost. If $i = 1$ and $B_j = 0$ then remove this server site and the request in I'. This changes neither the assignments of BODS to the other requests, nor the cost to BODS. Further, the optimal cost does not go up.

We now consider restriction (B). Let q be a leaf request in I that arrives at a site j and is serviced locally by BODS. We create I' by removing q from I, and decrementing B_j by 1. This does not change how BODS services the requests, nor the cost to BODS. This also doesn't change the cost in the optimal solution because we know that there are are at most B_j requests at site j in the input, and thus the optimal solution services q locally on input I.

We consider restriction (C). For any instance with fewer than B requests, the optimal solution must have servers left unassigned. For each unassigned server in the optimal solution, another request can arrive at its site. This does not change the cost of the optimal solution, and will not decrease the cost to BODS. ■

Theorem 3 *BODS is $+1$-server $O(\log k)$-competitive for online transportation on a star with k leaves.*

Proof: By Lemma 2, we need only consider restricted inputs. For notational convenience, we relabel the sites based on the number of servers at each site as well as tie-breaking order. In particular, we label the sites so that $B_1 \leq B_2 \leq ... \leq B_k$. Furthermore, we label them such that if $B_j = B_{j+1}$, then site j comes earlier than site $j + 1$ in the tie-breaking order. Let e_i be the number of servers that site i is augmented by in our online setting for BODS. Eventually we will set each e_i to one, but we believe that it is instructive to leave

the e_i's as variables in the proof. The only property that we need of the e_i's is that they are non-decreasing, that is, $e_i \leq e_{i+1}$.

We claim that the server sites die in order of increasing site number. To see why, remember that each request in a restricted input is remotely serviced by BODS. Thus the first requests must be root requests, and BODS will choose to assign servers from server sites in a round robin fashion. Root requests will continue until site 1 is dead. We know that site 1 will be the first to die since we have numbered the sites in non-decreasing order of the number of servers at each site, and in the case of sites with the same number of servers, we have taken care to assign lower numbers to sites that are earlier in BODS's tie-breaking order. After site 1 is dead, again since BODS services all requests remotely, all successive requests must arrive at either site 1 or the root until site 2 is dead, and so on.

Define *round* 0 to be the sequence of requests from the first request up until site 1 is dead. For $i \geq 1$, define round i to be the sequence of requests after round $i-1$ up until site $i+1$ is dead, or until a total of B requests have been made, whichever comes first. For $i \geq 0$, let r_i be the number of requests in round i. Note that round 0 consists only of root requests, whereas for $i \geq 1$, round i may consist of both root requests and leaf requests at dead sites.

Let m be the round in which the Bth request appears. Note that $1 \leq m \leq k-1$ since site $m+1$ dies in round m and there are only k server sites. For $j = 1 \ldots m$, let x_j be the number of root requests in round j. Note that $x_0 = r_0$ and $0 \leq x_i \leq r_i$ for $1 \leq i \leq m$. Recall that until the end of round i, requests are made only at the root and the first i dead sites. The number of root requests through round i is $\sum_{j=1}^{i} x_j$. Since the input is restricted, there are at most B_j requests on site j. Thus, the number of leaf requests through round i is at most $\sum_{j=1}^{i} B_j$. So for $1 \leq i \leq m$,

$$\sum_{j=1}^{i} r_j \leq \sum_{j=1}^{i} (B_j + x_j). \tag{2.1}$$

Note that $r_0 + \sum_{j=1}^{m} x_j$ is the number of total root requests in a restricted input. In a restricted input the optimal solution only pays for root requests. Hence,

$$C_{OPT}(I) = r_0 + \sum_{j=1}^{m} x_j.$$

Since BODS pays at most 2 to service a request, then

$$C_{BODS}(I) \leq 2B \leq 2 \left(r_0 + \sum_{j=1}^{m} x_j + \sum_{j=1}^{m} B_j \right).$$

We now formulate the number of requests in each round. Let α_1 be the position of site 1 in BODS's tie-breaking order on the sites 1 through k. Since no site has fewer than $B_1 + e_1$ servers, and there are k sites, there must be

$$r_0 = e_1 k + (B_1 - 1)k + \alpha_1$$

requests before site 1 is dead. To see this, note that a total of e_1 servers at each site are assigned after the first $e_1 k$ requests, and $B_1 - 1$ more servers are assigned from each site after the next $(B_1 - 1)k$ requests, and the $B_1 + e_1$th server of site 1 is assigned after another α_1 requests.

Similarly, since at the beginning of round i, only $k - i$ sites are alive, let $\alpha_{i+1} \leq k - i$ be the position of site $i+1$ in BODS's tie-breaking order on the remaining sites $i+1$ through k. The number of requests that are in round i is then

$$r_i = (k - i + 1 - \alpha_i) + (B_{i+1} + e_{i+1} - (B_i + e_i) - 1)(k - i) + \alpha_{i+1},$$

for $1 \leq i \leq m$. To see why, consider two cases.
Case 1 ($B_{i+1} > B_i$). We need $k - i + 1 - \alpha_i$ requests to finish clearing the $B_i + e_i$th server at each of the remaining $k - i + 1 - \alpha_i$ sites. We then need another $(B_{i+1} + e_{i+1} - (B_i + e_i) - 1)(k - i)$ requests to use up all but one server at the site(s) with $B_{i+1} + e_{i+1}$ servers. Finally, we need α_{i+1} more requests to reach the $B_{i+1} + e_{i+1}$th server of site $i + 1$ in the tie-breaking order.
Case 2 ($B_{i+1} = B_i$). The $k - i + 1 - \alpha_i$ requests will finish clearing off the $B_i + e_i = B_{i+1} + e_{i+1}$th server at the remaining $k - i + 1 - \alpha_i$ sites. Then reverting $k - i$ requests brings us to the point where only the first $B_i + e_i = B_{i+1} + e_{i+1}$th server at any site was used (this site may or may not be site 1). So we need α_{i+1} more requests which, by definition of α_{i+1}, bring us to the point where the $B_{i+1} + e_{i+1}$th server of site $i + 1$ is used.

Thus, for $1 \leq i \leq m$,

$$\sum_{j=1}^{i} r_j = \sum_{j=1}^{i} ((k - j + 1 - \alpha_j) + (B_{j+1} + e_{j+1} - (B_j + e_j) - 1)(k - j) + \alpha_{j+1})$$

$$= k - \alpha_1 + (B_{i+1} + e_{i+1})(k - i) - (B_1 + e_1)k + \sum_{j=1}^{i}(B_j + e_j) - (k - i) + \alpha_{i+1}.$$

Substituting into 2.1 then solving for B_{i+1} gives us that for all i where $1 \leq i \leq m$,

$$\begin{aligned}
B_{i+1} &\leq \frac{B_1 k + \sum_{j=1}^{i} x_j + \alpha_1 - \alpha_{i+1} - i}{k - i} + \frac{e_1 k - \sum_{j=1}^{i} e_j - e_{i+1}(k - i)}{k - i} \\
&\leq \frac{B_1 k + \sum_{j=1}^{i} x_j}{k - i} + 1 + \frac{e_1 k - \sum_{j=1}^{i} e_j - e_{i+1}(k - i)}{k - i} \quad \text{because } \alpha_1 \leq k \text{ and } \alpha_{i+1} \geq 1 \\
&\leq \frac{B_1 k + \sum_{j=1}^{i} x_j}{k - i} + 1 + e_1 - e_{i+1} \quad \text{because } e_1 \leq e_2 \leq \ldots \leq e_k \\
&\leq \frac{B_1 k + \sum_{j=1}^{i} x_j}{k - i} + 1. \qquad (2.2)
\end{aligned}$$

Recall that

$$\begin{aligned}
C_{BODS}(I) &\leq 2\left(r_0 + \sum_{j=1}^{m} x_j + \sum_{j=0}^{m-1} B_{j+1}\right), \\
&\leq 2\left(r_0 + \sum_{j=1}^{m} x_j + \sum_{j=0}^{m-1} \frac{B_1 k + \sum_{i=1}^{j} x_i}{k - j} + \sum_{j=0}^{m-1} 1\right) \quad \text{by (2.2)} \\
&\leq 2\left(r_0 + \sum_{j=1}^{m} x_j + \left(B_1 k + \sum_{j=1}^{m-1} x_j\right) \sum_{j=0}^{m-1} \frac{1}{k - j} + m\right) \\
&\leq 2\left(C_{OPT}(I) + C_{OPT}(I) \ln \frac{k}{k - m} + C_{OPT}(I)\right) \\
&\leq (2 \ln k + 4) C_{OPT}(I) \text{ since } m \leq k - 1.
\end{aligned}$$

Hence the result follows. ■

The following lemma will be useful in our analysis of BODS on HST's. The proof mimics the proof of Theorem 3.

Lemma 4 *In a star, the number of requests serviced remotely by BODS is at most* $2 \ln k + 4$ *times the number of requests serviced remotely by any optimal assignment with* B_j *servers per site.*

3 Generalization to HSTs

We now generalize this result to one on hierarchically separated trees. The theorem and proof presented in this section are based on that of section 3 in [12].

Definition 5 *An α-hierarchically separated tree (α-HST) is a rooted tree $T = (V, E)$ with a distance function on the edges such that*

1. *If two nodes are siblings in the tree, they are both the same distance from their parent.*
2. *The distance from a node to its parent is α times the distance of the node to its child.*
3. *All leaves are at the same level of the tree.*

To enable us to analyze BODS when the metric space is an HST, we define a variation of our original problem. The results we obtain on this variation will translate back to the original problem. Let our original transportation problem be referred to as TRN. Recall that in TRN the input request sequence was made up of at most B requests, one request for each server that OPT has. We now define the problem TRN2 to be the same as our original problem except for the following modifications. The input request sequence may now have up to $B + k$ requests (the number of servers available to BODS). At any time, to service a request, an algorithm may choose to pay a *service fee* instead of assigning a server to the request. The cost of the service fee is defined to be the length of the path from the request, to the root of the tree, to a leaf of the tree. For example, if a request appears at the root of the tree, the service fee is equal to the root to leaf distance in the HST. As another example, if a request appears at a leaf of the tree, the service fee is equal to twice the root to leaf distance in the HST.

We must now describe the algorithm BODS for this new problem. BODS will always assign a server to a request, never choosing to pay the service fee. To choose a server, BODS behaves exactly as described above.

Note that the service fee is always an upper bound on the cost of servicing a request using a server. The service fee is set intentionally high so as to deter the optimal offline algorithm from choosing to pay a service fee over choosing to assign a server to the request. Thus, we may assume that the optimal offline algorithm chooses to pay the service fee for a request only when it runs out of servers.

Lemma 6 *If BODS is $+1$-server c-competitive for TRN2, then BODS is $+1$-server c-competitive for TRN.*

Lemma 7 *There exists a worst-case input instance I in TRN2 against BODS in which OPT does not pay any service fees.*

Theorem 8 *BODS is +1-server $(8 \ln k + 10)$-competitive for TRN2 when the metric space is an α-HST T where $\alpha \geq 4 \ln k + 9$, the server sites are at the leaves of T, and the requests arrive at the leaves or at the root of T.*

Proof: By lemma 7, we need only consider input instances in $\mathcal{I}_{TRN2}$ where OPT does not pay any service fees, or in other words, where there are at most B requests.

The proof is by induction on the number of levels in T. The base case is the uniform star metric, which we already proved in Theorem 3. For the inductive step, we show that if the theorem is true for each subtree S_i of T that is rooted at child i of the root of T, $1 \leq i \leq z$, then it must be true for T itself.

To be more precise, let H be the height of T, let S be any α-HST of height $h \leq H - 1$ with server sites at the leaves, and let $C_{ALG}(S)$ be the cost of running the algorithm ALG on any input sequence of requests at the leaves or root of S. We assume that $C_{BODS}(S)/C_{OPT}(S) \leq 8 \ln k + 10$ holds for any S, and show that this assumption implies that for any α-HST T of height $h + 1$, $C_{BODS}(T)/C_{OPT}(T) \leq 8 \ln k + 10$.

Let δ be the distance from r, the root of T, to each of its children. Let $\beta = \sum_{i=1}^{H-1} 1/\alpha^i$, where H is the number of levels in T. Note that, $(\beta+1)\delta$ is the distance from r to a leaf of T and $\beta\delta$ is the distance from one of r's children to one of the leaves of T descendant from that child. Let m_i^* and m_i be the number of times that OPT and BODS (respectively) assigned servers in subtree S_i to requests that are *not* in S_i. Let $m^* = \sum_{i=1}^{z} m_i^*$ and $m = \sum_{i=1}^{z} m_i$. So m^* and m are the number of servers that OPT and BODS, respectively, assign to requests outside their subtrees. Let S_i^+ be the instance on subtree S_i defined by the servers of T in S_i, and a subsequence of the requests of the input sequence in T that consists of requests that are serviced by BODS using servers in S_i, where requests outside of S_i are replaced by requests at the root of S_i. Note that there are m_i replacements. Note that S_i is an α-HST with depth $H-1$. Let S_i^B be the instance on subtree S_i obtained from S_i^+ by removing the m_i requests at the root of S_i.

It is the case that

$$C_{BODS}(T) \leq \sum_{i=1}^{z} C_{BODS}(S_i^+) + \sum_{i=1}^{z}((\beta+1)\delta+\delta)m_i. \tag{3.3}$$

To justify this, first note that the second term of the right hand side reflects (for each subtree) m_i times the distance from the requests outside S_i (at the leaves of T) to the root r of T, then down to the root of S_i. Also, remember that by definition of BODS there are no requests for which BODS pays a service fee. Thus, any request x is assigned by BODS to a server y. If server y is in subtree S_i, then x belongs to S_i^+. If x appears in S_i, the cost of servicing x is accounted for in the i'th summand in the first summation. If x appears in another subtree or at the root of T, then the cost of servicing x is accounted for in the i'th summand in both the first and the second summations.

Let $R = 8\ln k + 10$. By the inductive hypothesis, we know that for all $1 \leq i \leq z$,

$$C_{BODS}(S_i^+) \leq R \cdot C_{OPT}(S_i^+). \tag{3.4}$$

It is the case that

$$C_{OPT}(S_i^+) = C_{OPT}(S_i^B) + \beta\delta m_i. \tag{3.5}$$

To see why, recall that the requests in S_i^+ are (by definition of S_i^B) the same as the requests in S_i^B with m_i requests added at the root. $\beta\delta$ is the distance from the root of S_i to a leaf in S_i. Note that OPT's cost for any root request in S_i^+ that it chooses to pay a service fee for is also $\beta\delta$. Thus, whether OPT services a root request by assigning a server to it or paying a service fee, the cost is at most $\beta\delta$. And OPT's cost on all leaf requests in S_i^+ will be the same as its cost (including any service fees) on S_i^B.

By (3.3), (3.4), (3.5), and the definition of m,

$$C_{BODS}(T) \leq R\sum_{i=1}^{z} C_{OPT}(S_i^B) + (R\beta\delta + 2\delta + \beta\delta)m. \tag{3.6}$$

Now all we have left to show is that the right hand side of (3.6) is less than or equal to $R \cdot C_{OPT}(T)$.

We start by observing that

$$C_{OPT}(T) \geq \sum_{i=1}^{z} C_{OPT}(S_i^B) + \delta m^*. \tag{3.7}$$

This inequality holds by the following reasoning. All requests serviced locally by OPT on S_i^B will be serviced locally by OPT on T in the same manner. The requests that OPT must pay a fee for in S_i^B must cost OPT on T a price of $2(\beta+1)\delta$ (the distance from a leaf up to the root and back down to a leaf), while only costing OPT on S_i^B a price of $2\beta\delta$ since S_i is one level shorter than T. This leaves at least an extra 2δ that OPT

has to pay for each of these requests on T. We also know that $OPT(T)$ pays $(\beta+1)\delta$ for each of the root requests that get counted into m^*. Both of these costs exceed δ and these are all the requests that comprise m^*.

Let $R_m = 2\ln k + 4$. By our assumption that $\alpha \geq 2R_m + 1$, we have

$$\beta = \sum_{i=1}^{h-1} 1/\alpha^i \leq \frac{1}{1-1/\alpha} - 1 = \frac{1}{\alpha - 1} \leq 1/(2R_m). \tag{3.8}$$

Note that $R = 4R_m + 2$. Using this fact along with 3.8, we have

$$R = \frac{1}{2}R + 2R_m + 1 \geq \frac{1}{2}R(2R_m\beta) + 2R_m + 2R_m\beta = R_m(R\beta + 2 + 2\beta) \geq R_m(R\beta + 2 + \beta). \tag{3.9}$$

We now need to relate the values m and m^*. Recall that BODS always services requests as locally as possible. If we think of a request that is matched to a server within its own subtree as a *local* assignment, we can reduce this input instance on T to one on the uniform star, where the subtrees of T are the leaves of the star. Since all inputs under consideration have at most B servers, this reduction will be an instance of the problem from section 2. Therefore by Lemma 4, we know that

$$m \leq R_m \cdot m^*. \tag{3.10}$$

Finally, by combining 3.6, 3.7, 3.9, and 3.10, we have $C_{BODS}(T) \leq R \cdot C_{OPT}(T)$. ■

Combining Theorem 8 with Lemma 6 immediately gives us the following result.

Theorem 9 *BODS is +1-server $(8\ln k + 10)$-competitive for TRN when the metric space is an α-HST T where $\alpha \geq 4\ln k + 9$, the server sites are at the leaves of the tree, and the requests arrive at the leaves of T.*

4 Generalization to any metric space

We are now ready to extend our results to any metric space that has k server sites and plus one server per site for the online algorithm. In the general metric, the requests are no longer restricted to arriving only at existing server sites. Requests can now arrive at any point in the metric space.

We now present GBODS, an algorithm for the online transportation problem in any given metric space. The new algorithm is as follows: first use the procedure given in [4] to generate a random $(9 + 4\ln k)$-HST, call it T, from the metric induced on the k server sites. Then, whenever a request q arrives in the original metric space, we find its nearest server site, and create a new request there, calling it q'. Let $Q = \{q_1, q_2, \ldots, q_B\}$ be the sequence of requests that arrive, and let $Q' = \{q'_2, q'_3, \ldots, q'_B\}$ be the corresponding set of requests created at the server sites nearest to the requests in Q. Let $T(Q')$ be the input instance on HST T with request sequence Q'. We use BODS on $T(Q')$ to find an available server s'_i for each q'_i in Q'. We then assign each server s'_i to each original request q_i in Q.

Fakcharoenphol et al [4] showed that any metric space of n points can be approximated by a randomly generated α-HST where the points in the metric will be at the leaves of the tree, and the expected distance between points in the tree will be no more than $\alpha \log n$ times their original distance. Applying this fact along with our Theorem 9, and losing a constant factor due to applications of the triangle inequality when mapping the solution of input $T(Q')$ back to the points of Q in the original metric space, we have the following theorem.

Theorem 10 *In expectation, the algorithm GBODS is +1-server $O(\log^3 k)$-competitive for the online transportation problem.*

Acknowledgments: We thank Anupam Gupta for helpful discussions about online matching and metric embedding techniques.

References

[1] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *ACM Symposium on Theory of Computing*, pages 161–168, 1998.

[2] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Symposium on Foundations of Computer Science*, page 379, 1998.

[3] Béla Csaba and András Pluhar. A randomized algorithm for the on-line weighted bipartite matching problem. Submitted for publication.

[4] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *ACM Symposium on Theory of Computing*, pages 448–455, 2003.

[5] Anupam Gupta, Mohammad T. Hajiaghayi, and Harald Räcke. Oblivious network design. In *ACM-SIAM Symposium on Discrete algorithm*, pages 970–979, 2006.

[6] Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[7] Bala Kalyanasundaram and Kirk Pruhs. On-line network optimization problems. In *Developments from a June 1996 seminar on Online algorithms*, pages 268–280. Springer-Verlag, 1998.

[8] Bala Kalyanasundaram and Kirk R. Pruhs. The online transportation problem. *SIAM Journal of Discrete Mathematics*, 13(3):370–383, 2000.

[9] Jeff L. Kennington and Richard V. Helgason. *Algorithms for Network Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1980.

[10] Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

[11] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart & Winston, New York, 1976.

[12] Adam Meyerson, Akash Nanavati, and Laura Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *ACM-SIAM Symposium on Discrete algorithms*, pages 954–959, 2006.