Online Bottleneck Matching

Barbara M. Anthony[1] and Christine Chung[2]

[1] Mathematics and Computer Science Department, Southwestern University, Georgetown, TX
anthonyb@southwestern.edu
[2] Department of Computer Science, Connecticut College, New London, CT
cchung@conncoll.edu

Abstract. We consider the online bottleneck matching problem, where k server-vertices lie in a metric space and k request-vertices that arrive over time each must immediately be permanently assigned to a server-vertex. The goal is to minimize the maximum distance between any request and its server. Because no algorithm can have a competitive ratio better than $O(k)$ for this problem, we use resource augmentation analysis to examine the performance of three algorithms: the naive GREEDY algorithm, PERMUTATION, and BALANCE. We show that while the competitive ratio of GREEDY improves from exponential (when each server-vertex has one server) to linear (when each server-vertex has two servers), the competitive ratio of PERMUTATION remains linear when an extra server is introduced at each server-vertex. The competitive ratio of BALANCE is also linear with an extra server at each server-vertex, even though it has been shown that an extra server makes it constant-competitive for the min-weight matching problem.

1 Introduction

We consider the online bottleneck matching problem, where we are given k server-vertices located in a metric space, and k request-vertices that arrive over time. As each request-vertex arrives, it must be immediately and permanently matched to a server-vertex. Our goal is to minimize the maximum distance between any request-vertex and its assigned server-vertex.

The standard technique for studying algorithms for online problems is competitive analysis. The *competitive ratio* of an algorithm is the worst-case ratio of the cost of the algorithm's solution to the cost of the optimal offline solution (which knows all request locations in advance). Kalyansundaram and Pruhs [4] proposed an algorithm, PERMUTATION, in the context of the corresponding online *min-weight* matching problem, where the goal is to minimize the total (or average) distance between request-vertices and server-vertices. Without proof, [4] mentioned that PERMUTATION achieves a competitive ratio of $2k - 1$ for the online bottleneck matching problem. Idury and Schäffer [3] then proved that no algorithm can achieve a competitive ratio better than approximately $1.5k$. The basic GREEDY algorithm, which assigns each arriving request to the nearest available server-vertex, has a competitive ratio that is $\Omega(2^k)$ (see Section 2).

The prohibitive general lower bound on the problem and the exceedingly poor performance of a simple and natural algorithm like GREEDY motivate us to consider a benchmark that is less formidable than the optimal solution, in order to attain a more

informative analysis of these algorithms. Specifically, we employ a *weak adversary model* of analysis in pursuit of further insight on the performance of these (and related) algorithms for the bottleneck matching problem. The weak adversary, or *resource augmentation*, model of analysis has long been used effectively in the study of matching and scheduling problems (e.g., [5, 6, 8, 1]). Results obtained under this model can be viewed as "bicriteria" results, which have also become an informative and successful approach in other sub-fields of algorithms (e.g., [9, 2]).

In our setting with resource augmentation, we ask how well the online algorithm performs when it has multiple servers (namely two) per server-vertex, while the optimal offline solution only has one; thus the online algorithm can service twice as many request-vertices with each server-vertex. Following [5], we will use the term *halfOPT-competitive ratio* to refer to the competitive ratio of an online algorithm with server-vertices that have two servers when compared with an optimal offline solution with each server-vertex having a single server.

Resource augmentation was used to study the corresponding online min-weight matching problem in [5]. They showed that by having two servers per server-vertex, the competitive ratio of GREEDY improves from $\Theta(2^k)$ to a halfOPT-competitive ratio of $\Theta(\log k)$. They then proposed an algorithm BALANCE, which is a modified form of GREEDY that is more judicious in its use of the additional server at each server-vertex. They show that BALANCE has a halfOPT-competitive ratio of $O(1)$.

Our results for the online bottleneck matching problem for $k \geq 2$ are as follows. (Naturally, when there is a single request-vertex and server-vertex ($k = 1$) the algorithms all perform optimally.)

1. GREEDY has a competitive ratio of at least 2^{k-1}, and at most $k2^{k-1}$.
2. PERMUTATION (proposed in [4] and [7]) is $(2k-1)$-competitive, and this is tight. This is comparable to its performance for the min-weight objective, for which it is also $(2k-1)$-competitive. This $O(k)$ upper bound on the ratio is asymptotically tight with the $\Omega(k)$ general lower bound for the problem of [3].
3. GREEDY has a halfOPT-competitive ratio of no more than $(k-1)$. Note that this is an exponential improvement in competitive ratio from simply having two servers available per server-vertex.
4. GREEDY has a halfOPT-competitive ratio of at least $(k+1)/2$. Interestingly, this is still exponentially worse than its performance for the corresponding min-weight problem, where it has a halfOPT-competitive ratio of $2\log k$ [5].
5. BALANCE (proposed in [5]), a modified form of GREEDY designed for the setting of multiple servers per server-vertex, has a halfOPT-competitive ratio of $k-1$.
6. BALANCE has a halfOPT-competitive ratio of at least $(\frac{1}{c}+1)^{\log(k+1)-1} = \Omega(k)$. This is in contrast with the fact that BALANCE has a halfOPT-competitive ratio of $O(1)$ for the corresponding min-weight problem [5].
7. PERMUTATION has a halfOPT-competitive ratio of k and this is tight. (Note that having two servers per server-vertex does not improve PERMUTATION's asymptotic performance guarantee, as it did so dramatically with GREEDY.)

A table summarizing these and related results is shown below.

Table 1. Lower bounds and upper bounds for the various algorithms. All bottleneck objective results are from the present work, though the PERMUTATION bounds without resource augmentation were hinted at in [4]. The result marked by † is immediate from the corresponding bound without resource augmentation. BALANCE is only defined in the resource augmentation setting.

Objective	Adversary	Algorithm					
		GREEDY		PERMUTATION		BALANCE	
		LB	UB	LB	UB	LB	UB
min-bottleneck	OPT	2^{k-1}	$k2^{k-1}$	$2k-1$	$2k-1$	N/A	N/A
	halfOPT	$(k+1)/2$	$k-1$	k	k	$\Omega(k)$	$k-1$
min-weight	OPT [4]	2^k-1	2^k-1	$2k-1$	$2k-1$	N/A	N/A
	halfOPT	$\Theta(\log k)$ [5]		$O(1)$	$2k-1^{\dagger}$	$\Theta(1)$ [5]	

While resource augmentation has the potential to improve the competitive ratio, these results suggest that in some sense the bottleneck objective is more difficult than the total distance objective. Resource augmentation greatly helps GREEDY for the minimum weight objective, but none of the three algorithms break the $\Omega(k)$ barrier for the bottleneck objective. Perhaps this can be explained by noting that for the minimum weight objective, any sub-optimal assignment is mitigated by the total cost, whereas with the bottleneck objective, a poor assignment can dominate, even with resource augmentation. Our results suggest that GREEDY can be a reasonable choice of algorithm for the bottleneck objective with resource augmentation, due to its relative simplicity, and comparable performance to BALANCE and PERMUTATION, despite its decay in performance as its adversary gets stronger.

Section 2 provides some results for the algorithms without resource augmentation. We then consider three algorithms with resource augmentation: GREEDY (Section 3), BALANCE (Section 4), and PERMUTATION (Section 5).

2 Preliminaries

Formally, the online bottleneck matching problem is as follows: Given a collection $S = \{s_1, s_2, \ldots, s_k\}$ of server-vertices in a metric space M, the online algorithm A sees over time a sequence of request-vertices $R = \{r_1, r_2, \ldots, r_k\}$ also in M. When request-vertex r_i arrives, algorithm A must assign a server-vertex $s_{\sigma(i)}$ to service that request, with cost equaling the distance $d(r_i, s_{\sigma(i)})$ (we use the terms cost and distance interchangeably). Once an assignment is made, it cannot be changed. While A does not know the sequence of requests in advance, its goal is to minimize the bottleneck distance of the overall assignment, that is minimize $\max_i \; d(r_i, s_{\sigma(i)})$. We refer to the assignment (or "matching") that optimizes this objective as OPT. As is typical of online problems, we use competitive analysis, and seek to minimize the worst-case ratio of the online bottleneck cost to the optimal (offline) bottleneck cost. An online algorithm is α-competitive if this ratio is at most α for all possible instances. We use $cost(\cdot)$ to represent the bottleneck weight of a particular assignment, e.g. $cost$(OPT). Throughout

the paper, $\epsilon > 0$ represents an arbitrarily small constant, typically used to break ties when assigning requests to servers.

We now prove a few basic results about the online bottleneck matching problem without resource augmentation that have been hinted at in the existing literature (e.g., see the Conclusion of [4]). We consider both the standard GREEDY algorithm, as well as PERMUTATION, introduced by Kalyanasundaram and Pruhs (a similar algorithm was also studied by [7]). Note that the algorithm BALANCE is only defined when there are multiple servers per server-vertex.

2.1 Analysis of GREEDY

As its name suggests, GREEDY assigns the nearest available server at a server-vertex to each request-vertex as it arrives. While this algorithm can perform well on some instances, GREEDY is exponentially bad against OPT. In fact, this can be exhibited by the same instance of [4] that demonstrates GREEDY is exponentially bad against OPT for the corresponding objective of minimizing total weight.

Theorem 1. *The competitive ratio of* GREEDY *is at least* 2^{k-1} *for the bottleneck matching problem.*

Proof. Let M be a subspace of the real line, with the standard distance metric. Set $s_1 = -1 - \epsilon$ and $s_i = 2^{i-1} - 1$ for $2 \leq i \leq k$. Let $r_i = 2^{i-1} - 1$ for $1 \leq i \leq k$. GREEDY assigns request r_i to s_{i+1} for $i < k$ (as the request-vertices and server-vertices are collocated), and then must assign r_k to s_1, for a bottleneck cost of $2^{k-1} + \epsilon$. OPT, however, matches each r_i to the corresponding s_i, giving $cost$(OPT) $= 1 + \epsilon$.

Theorem 2. *The competitive ratio of* GREEDY *is at most* $k2^{k-1}$ *for the bottleneck matching problem.*

Proof. Let N_i be the partial matching constructed by GREEDY after i requests have been revealed. Let w_i be the cost of the bottleneck edge in N_i. (Ties do not matter, as the concern is the cost, not the particular bottleneck edge.) Let b_i be the cost of the bottleneck edge in M_i. We can assume, without loss of generality, by renumbering the vertices that GREEDY services r_i with s_i. We prove inductively that $w_i \leq i2^{i-1}b_i$. For $i = 1$, $M_1 = N_1$ and the result follows. Now assume that the result holds for $i - 1$, and we verify that it holds for i.

If the weight of the edge (r_i, s_i) selected by GREEDY to service r_i is at most w_{i-1}, then we are done. By [4], the weight of (r_i, s_i) is at most the sum of the weights of the edges in M_i and the edges in N_{i-1}, and the sum of the weights in N_{i-1} is at most $2^{i-1} - 1$ times the sum of the weights in M_{i-1}. Thus w_i is at most 2^{i-1} times the sum of the weights in M_i, as the minimum weight matching can only increase with an additional request. Noting that the sum of the weights in M_i is at most the number of edges in M_i times the weight of the most expensive (i.e. bottleneck) edge gives that $w_i \leq i2^{i-1}b_i$. Since this holds for all i, and $b_k = OPT$, the result holds.

2.2 Analysis of PERMUTATION

Informally, PERMUTATION assigns requests as follows. Note that the assignment of request-vertices to server-vertices is a matching. To choose a server for request r_i, consider the optimal matching of the first i requests, and the optimal matching of the first $i-1$ requests. There is exactly one server that is matched in the former scenario and not in the latter. PERMUTATION matches that server to the current request r_i. Observe that PERMUTATION guarantees that if a request arrives at an unused server-vertex, it is matched to the server at that server-vertex.

More formally, as defined in [4], let $R_i \subset R$ be the first i request-vertices. A *partial matching* of R_i is a perfect matching of R_i with a subset of the servers of S. Let M_0 and P_0 be empty. Define M_i to be the edges that form a minimal weight partial matching on R_i where the number of edges in $M_i - M_{i-1}$ is minimized, choosing arbitrarily if multiple such matchings exist. Let $S_i \subset S$ be the server-vertices incident to an edge in M_i. Let P_i denote the partial matching constructed by PERMUTATION after the first i requests. PERMUTATION constructs P_{i+1} by computing M_{i+1}, assigning r_{i+1} to the unique server-vertex $s \in S_{i+1} - S_i$, and adding that edge to the matching P_i.

We now show that PERMUTATION is $(2k-1)$-competitive, which was stated without proof in the Conclusion of the preliminary version of [4]. The proof is similar to the proof in [4] which shows that PERMUTATION is $(2k-1)$-competitive for the online minimum-weight matching problem.

Theorem 3. PERMUTATION *is* $(2k-1)$*-competitive for the bottleneck matching problem.*

Proof. We prove inductively that $cost(P_i)$ is at most $2i-1$ times the $cost(M_i)$. Clearly, $P_1 = M_1$, and the inequality holds. Assume that the inductive hypothesis holds for $i-1$, that is, $cost(P_{i-1}) \leq (2(i-1)-1) \cdot cost(M_{i-1})$.

Assume PERMUTATION services request r_i with server-vertex s_j. Consider the bottleneck distance of P_i. By construction, $P_i = P_{i-1} \cup r_i s_j$. Thus, $cost(P_i)$ is the maximum of $cost(P_{i-1})$ and $d(r_i, s_j)$. Note also that $cost(M_{i-1})$ is at most $cost(M_i)$. Thus, if the $cost(P_i)$ is $cost(P_{i-1})$, then by induction it is at most $(2(i-1)-1) \cdot cost(M_{i-1})$, which is at most $(2i-1) \cdot cost(M_i)$. Otherwise, the $cost(P_i)$ is $d(r_i, s_j)$.

Let M' be the union of the matching M_{i-1} and the edge $r_i s_j$. Let H be $M_i \oplus M'$. I.e., let H be the set of all edges that are in exactly one of M_i and M'. Intuitively, H captures the cascading effect of reassignments upon the arrival of the latest request r_i. (H appears again in Section 5, where it is discussed further.) H consists of one alternating cycle (possibly empty). By the triangle inequality, we have that $d(r_i, s_j)$ is at most the total cost of the edges in H, less itself. Thus, $d(r_i, s_j)$ is at most the weight of M_{i-1} (defined as the sum of the costs of the edges in M_{i-1}) plus the weight of M_i. Recall that $cost(M_{i-1}) \leq cost(M_i)$. Furthermore, since the bottleneck distance is the largest edge in the matching, the sum of the weights of the edges in the matching is at most the number of edges times the bottleneck weight. Thus, $d(r_i, s_j)$ is at most $(2i-1) \cdot cost(M_i)$, completing the proof.

Theorem 4. *The competitive ratio of* PERMUTATION *is at least* $2k-1$ *for the bottleneck matching problem.*

Proof. Let M be a subspace of the real line, with the standard distance metric. Set $s_i = i$ for $1 \leq i \leq k$. Let $r_i = i + .5 + \epsilon$ for $1 \leq i \leq k$. PERMUTATION matches r_i to s_{i+1} when it exists, and matches the final request, r_k, to s_1, for a bottleneck cost of $k - .5 + \epsilon$. OPT assigns r_i to s_i, so all edges have a cost of $.5 + \epsilon$, which is thus $cost$(OPT). Thus, the performance on this instance is $(2k - 1 + 2\epsilon)/(1 + 2\epsilon)$, which approaches $2k - 1$. (The ϵ could be removed if ties can be broken arbitrarily.)

Resource augmentation was used in [5] to show that, for the min-weight objective, GREEDY has a halfOPT-competitive ratio of $O(\log k)$, in contrast with its $\Omega(2^k)$ competitive ratio without resource augmentation. Motivated in part by these results, we turn to a resource augmentation setting for the bottleneck objective.

3 Bicriteria analysis of Greedy

Noting that "the poor competitive ratio of an intuitive greedy algorithm may not reflect the fact that it may perform reasonably well on 'normal' inputs", [5] adopts a *weak adversary model*, in which the adversary has fewer resources than the online algorithm. Their work address the online transportation problem, which is a generalization of the min-weight matching problem. We perform a similar analysis for the bottleneck matching problem, and show that the improvement for GREEDY is more limited for our objective.

While each server-vertex in OPT can service exactly one request, the online algorithm can assign requests to two servers at each server-vertex. Thus, as in [5] we say that the *halfOPT-competitive ratio* of an online algorithm A is the supremum over all instances I with at most k requests of $A(I)/OPT(I)$ where A has two servers available at each server-vertex, while OPT only has one.

We now show that the halfOPT-competitive ratio for GREEDY is linear in the number of requests. Since each server-vertex s_i has two servers in the online setting, we denote them by s_i^1 and s_i^2 as needed. Without loss of generality, we assume that s_i^2 is not used unless s_i^1 is already in use. The adversary has only s_i^1 available to it. We first prove a lemma about the *response graph* G, defined in [5] to be $G = (S \cup R, E)$, where E is the set of edges that includes the online edge $(r_i, s_{\sigma(i)})$ and adversary edge (r_i, s_i) for each request r_i.

Lemma 1. *Each connected component of G contains exactly one cycle.*

Proof. By Lemma 1 of [5], there is at most one cycle in each connected component of the response graph G. So it remains to show that there is at least one cycle in each connected component of G. To do this, we will show that a connected component C of G cannot be a tree. Since a tree must have one more vertex than edges, it suffices to show that C has an equal number of edges and vertices. We first observe that, because OPT is a perfect bipartite matching between server-vertices and request-vertices in G, any connected component of G must have an equal number of server-vertices and request-vertices. (Otherwise, some connected component would have one fewer server than request, and OPT would not be able to match that extra request to any server.) Hence the number of vertices in C is $2 \cdot R_C$, where R_C is the number of request vertices in

C. Next we observe that C must also have $2 \cdot R_C$ edges since each request-vertex in G must have exactly two incident edges (one from the online algorithm and one from OPT). We have now shown that C has an equal number of vertices and edges.

Theorem 5. *The halfOPT-competitive ratio of* GREEDY *for the bottleneck matching problem is at most* $k-1$ *for* $k \geq 2$ *server-vertices.*

Proof. Let (r_i, s_j) be the online bottleneck edge in the response graph, G. (If there are multiple edges with the maximum bottleneck cost, pick one arbitrarily.) Let (r_i, s_i) be the edge in OPT that serves request r_i. If $s_i = s_j$ then we're done. So we only consider the case that $s_i \neq s_j$. Now consider the connected component containing r_i. By Lemma 1 this connected component has exactly one cycle. Note that this cycle may have trees joined to it at the vertices on the cycle. Observe that all such junctions must represent a server-vertex, since each request can have at most two incident edges in the response graph, one for the online edge and one for the optimal edge. Consider separately the cases when r_i lies on the cycle, and when it does not.

If r_i is a vertex on the cycle, then since only server-vertices can be junctions, both the online and offline edges incident on r_i must lie on the cycle. Removing the online edge (r_i, s_j) from the cycle yields a tree which can be rooted at r_i. Since there are k request-vertices and k server-vertices, there are at most $2k$ vertices in the tree. Furthermore, the tree contains alternating levels of server-vertices and request-vertices. Each request-vertex has one child (the server-vertex chosen for it by OPT), and each server-vertex can have up to two children (the online edges).

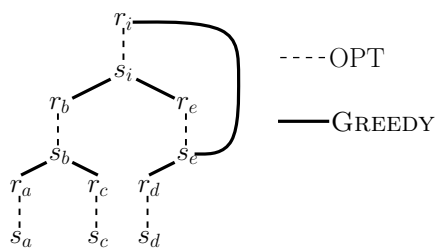

Fig. 1. An example response graph.

To upper bound the cost of the edge (r_i, s_j), it suffices to upper bound the distance of the shortest path from r_i to some server-vertex s_a with s_a^2 unused, since GREEDY picked s_j instead of s_a. Since r_i is the root of the tree, it suffices to find the cost of a path requiring the minimum number of edges that must be traversed to arrive at a leaf. Consider a version of the tree where the edges from a request to its child are contracted, thus resulting in a binary tree T with at most k vertices. Let $k_T \leq k$ refer to the number of vertices in the contracted tree. Since a full binary tree would have $\log(k_T)$ levels, a leaf of T, which may or may not be full, is reachable in at most $\log(k_T)$ edges. Uncontracting the edges (at most one per server-vertex) indicates that in the original

graph, there are at most $\log(k_T)$ optimal and $\log(k_T) - 1$ online edges between the root and some leaf, call it s_a.

Now consider the cost of the path in the tree from r_i to server-vertex s_a. By definition, any edge used in OPT must have cost at most $cost(OPT)$. Since all leaves of the tree are incident only with one edge, an OPT edge, the edge (r_a, s_a) is an edge in OPT, and thus has cost at most $cost(OPT)$. Proceeding from s_a to the root, the next edge on the path is an online edge, call it (r_a, s_b). GREEDY chose to assign r_a to s_b rather than s_a which had a server available, and thus has cost at most the cost of the edge from (r_a, s_a), which is again at most $cost(OPT)$. The next edge in the path, (r_b, s_b) is an edge in OPT, and thus has cost at most $cost(OPT)$. The next edge, the online edge (r_b, s_c) again was again chosen by GREEDY over the edge (r_b, s_a) and thus has cost at most the distance in the tree from r_b to s_a, which is bounded by the three edges previously mentioned in the path, for a total cost of at most $3 \cdot cost(OPT)$. This process continues, with successive edges in OPT having cost at most $cost(OPT)$ and successive online edges having cost at most $(2^h - 1) \cdot cost(OPT)$ where h represents the height of the request in the tree with the online edges contracted. As the edge incident to r_i in the subtree is an edge in OPT, the final edge in the path from s_a to r_i has cost at most $cost(OPT)$. Thus, the total cost of the path is at most $cost(OPT)$ for each of the $\log(k)$ edges in OPT and $\sum_{h=1}^{\log(k)-1}(2^h - 1) \cdot cost(OPT)$ for the online edges, giving $\sum_{h=0}^{\log(k-1)} 2^h \cdot cost(OPT) = (2^{\log(k)} - 1) \cdot cost(OPT) = (k-1) \cdot cost(OPT)$. Hence, since GREEDY assigned r_i to s_j instead of s_a, the online bottleneck edge cost is at most $(k-1) \cdot cost(OPT)$.

Now consider the case where r_i does not lie on the cycle. Removing (r_i, s_j) from the response graph partitions the original connected component into two connected components, with r_i and the original cycle now in separate connected components. As the original connected component contained exactly one cycle, the connected component rooted at r_i is a tree. By the same process, the upper bound on the distance from r_i to some leaf server-vertex s_a is at most $(k-1) \cdot cost(OPT)$, completing the proof.

The example used in [5] to provide a lower bound for GREEDY for the online transportation problem gives a lower bound of $k/2$ for GREEDY in this setting. We prove a slightly improved lower bound of $(k+1)/2$ in Corollary 1 in Section 4.

4 Bicriteria analysis of BALANCE

In this section we consider the BALANCE algorithm detailed in [5]. We first define some convenient notation for our resource augmentation model. As in the previous section, each server-vertex s_i in S is said to have a primary server s_i^1 and a secondary server s_i^2. Thus, while there are k vertices in S, one for each request in R, the online algorithm effectively has $2k$ servers to choose from. For BALANCE, the *pseudo-distance* from a request r_i to a primary server s_j^1 is the actual distance $d(r_i, s_j)$, while the *pseudo-distance* from the same request r_i to the secondary server s_j^2 is $c \cdot d(r_i, s_j)$, for a constant $c > 1$. (In [5], a $c > 11$ was specified.) BALANCE then uses GREEDY to assign arriving requests to servers, based on their pseudo-distances. (Thus BALANCE with $c = 1$ is

precisely GREEDY.) Note also that BALANCE only applies in the resource augmentation setting because it uses primary and secondary servers explicitly.

We begin with a lower bound on the halfOPT-competitive ratio of BALANCE.

Theorem 6. *The halfOPT-competitive ratio of* BALANCE *for the bottleneck matching problem is at least* $(\frac{1}{c}+1)^{\log(k+1)-1} = \Omega(k)$*, where* k *is the number of requests and* c *is the constant in the definition of* BALANCE.

Proof. Consider the following example on the line, where at each location the number of requests and server-vertices are powers of two. Let $L_0, L_1, L_2, \ldots, L_m$ be the $m+1$ server-vertex locations, where L_i has 2^{m-i} server-vertices. Similarly, the $m+1$ request locations are $R_0, R_1, R_2, \ldots, R_m$ where R_i has 2^{m-i} requests. Let $L_0 = -c$, $R_0 = 0$, and for $1 \leq i \leq m$, $L_i = R_i$.

We now determine the most extreme placement for the server-vertices so that OPT will assign requests at R_i to servers at L_i but that BALANCE will choose not to send any requests to L_0 until the final request. Thus OPT will have a bottleneck cost of c while BALANCE will pay c plus the location of the final server. Since c is fixed, the ratio will grow with the location L_m.

We break ties at our convenience. (Alternatively, a small $\epsilon > 0$ could be used to perturb the locations slightly to enforce such choices.) L_1 must be at 1 so that the secondary servers at L_1 (with a cost of $c \cdot 1$) are equally desirable as the primary servers at L_0 (cost of c) for the requests at R_0. L_2 must be chosen so that the requests at R_1 consider the secondary servers at L_2 (with cost $c \cdot d(L_1, L_2)$) as desirable as the primary servers at $L_0 = -c$, with cost $c+1$. Thus, $d(L_1, L_2) = \frac{c+1}{c}$, placing L_2 at $2 + \frac{1}{c}$. Repeating this process, L_i can be placed at $\sum_{j=1}^{i} \binom{i}{j} \frac{1}{c^{j-1}}$ for all $1 \leq i \leq m$.

We now find a closed form for the location of server L_m.

$$L_m = \sum_{j=1}^{m} \binom{m}{j} \frac{1}{c^{j-1}} = c \sum_{j=1}^{m} \binom{m}{j} \frac{1}{c^j} = c \left(\sum_{j=0}^{m} \binom{m}{j} \frac{1}{c^j} \right) - c \binom{m}{0} \frac{1}{c^0}.$$

Using the binomial theorem on the summation gives the expression $c(\frac{1}{c}+1)^m - c$. Thus, if L_m is the rightmost server, the bottleneck distance from L_0 to L_m is $c(\frac{1}{c}+1)^m$.

Note that the total number of requests is $k = \sum_{i=0}^{m} 2^i = 2^{m+1} - 1$. Thus $m = \log(k+1) - 1$. Thus the bottleneck cost for BALANCE is $c(\frac{1}{c}+1)^{\log(k+1)-1}$ where k is the number of servers/requests, and the bottleneck cost for OPT is c. If c is a fixed constant, then the lower bound on the competitive ratio is $(\frac{1}{c}+1)^{\log(k+1)-1}$.

Corollary 1. *The halfOPT-competitive ratio of* GREEDY *for the bottleneck matching problem is at least* $\frac{k+1}{2}$*, where* k *is the number of servers.*

Proof. Noting that $c = 1$ is precisely GREEDY, observe that if $c = 1$ this gives a competitive ratio of $2^{\log(k+1)-1} = \frac{k+1}{2}$.

We now show that the upper bound on the halfOPT-competitive ratio of BALANCE is a matching $O(k)$.

Theorem 7. BALANCE *has a halfOPT-competitive ratio of* k *for the bottleneck matching problem.*

Proof. The same argument as for the GREEDY upper bound (Theorem 5) applies. Note that it holds because the server-vertex s_a used in the argument is a leaf of the tree, which means the online algorithm has not used either of its servers. Thus the pseudo-distance to that vertex in BALANCE is the same as the original distance in GREEDY.

5 Bicriteria analysis of PERMUTATION

We next consider PERMUTATION with resource augmentation. As before, each server-vertex s_i has two servers in the online setting, the primary server s_i^1 and the secondary server s_i^2. Without loss of generality, we assume that a secondary server can only be used if the corresponding primary server is used. Again, we compare PERMUTATION to OPT which can serve exactly one request per server-vertex.

We now note how the definition of PERMUTATION from Section 2.2 applies to the resource augmentation setting. Let S^{aug} be the set of $2k$ servers available to the online algorithm. Then a *partial matching* of the first i requests is a perfect matching of these requests with a subset of S^{aug}. Define M_i to be the set of edges in a minimal weight partial matching of the first i requests that is "most similar" to M_{i-1}, in the sense that the number of edges in $M_i - M_{i-1}$ is minimized. Let $S_i \subset S^{aug}$ be the set of servers incident to an edge in M_i. By convention, M_0 is empty.

Suppose that PERMUTATION services request r_i with a server s_j^x at vertex s_j. Then define M' to be the union of M_{i-1} with the edge (r_i, s_j^x). Let P_i denote the partial matching constructed by PERMUTATION for the first i requests.

Intuitively, it may seem that PERMUTATION should benefit substantially from resource augmentation; the availability of a secondary server seemingly allows the algorithm to 'correct' itself if a request arrives and finds that the primary server it would have used in OPT was already in use. Yet, PERMUTATION has a halfOPT-competitive ratio of k and this is tight, as illustrated by the following lower bound instance and a matching upper bound guarantee. This is in comparison with its competitive ratio of $2k - 1$ in the absence of resource augmentation.

Theorem 8. PERMUTATION *has a halfOPT-competitive ratio of $\Omega(k)$ for the bottleneck matching problem.*

Proof. Fix a small constant $\epsilon > 0$. Without loss of generality, let k be odd. Consider the following instance, as depicted in Figure 2 for $k = 9$. Server vertices and requests s_i, r_i for $1 \leq i \leq k$ with i odd are placed along the line, in the order $s_1, r_1, s_3, r_3, \ldots, s_k, r_k$ where the distance between s_i and r_i is $1 + \epsilon$, and the distance between r_i and s_{i+2} is 1. For each $i \geq 3$, let request r_{i-1} be 1 away from s_i, and let server-vertex s_{i-1} be at a distance of $1 + 2\epsilon$ from r_{i-1}. All other distances are additive based on this graph.

Since PERMUTATION assigns requests based on M_i, note that M_1 assigns r_1 to s_3^1. Thus, PERMUTATION does the same. In M_2, this assignment remains, and r_2 is assigned to s_3^2, and again PERMUTATION behaves identically. In general, M_j for $j < k$ behaves as follows: if j is odd, r_j is assigned to s_{j+2}^1 and if j is even, r_j is assigned to s_{j+1}^2. PERMUTATION's assignments are identically M_j for $j < k$. Naturally, this pattern cannot continue for request r_k; observe that M_k that shares only about half of its edges with M_{k-1}. In particular, M_k assigns r_i to s_i^1 for i odd, and assigns r_j to s_{j+1}^2

for j even. Thus, PERMUTATION assigns the final request r_k to the only server used in M_k that was not used in M_{k-1}, that is, s_1^1. Hence, PERMUTATION assigns r_k to s_1, for a bottleneck cost of $k + \frac{k+1}{2}\epsilon$ (its other assignments all have cost 1).

Observe that OPT matches each r_i to its corresponding s_i, for a bottleneck cost of $1 + 2\epsilon$. Hence, PERMUTATION has a halfOPT-competitive ratio of $\Omega(k)$.

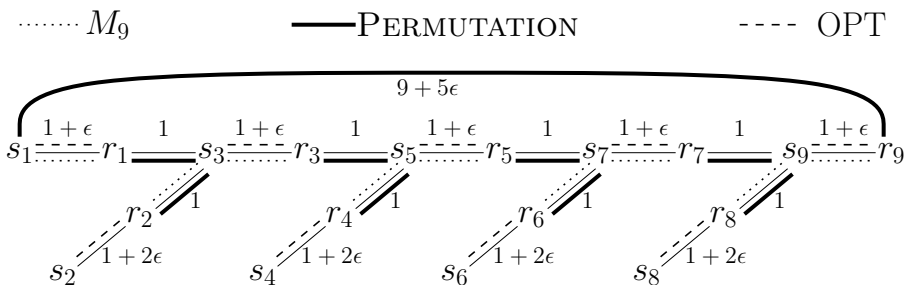

Fig. 2. Even with resource augmentation, PERMUTATION's cost can still be $k \cdot cost$(OPT).

We now develop a sequence of lemmas which show that $cost$(PERMUTATION) is at most $O(k) \cdot cost(OPT)$ for any instance. As in [4], let $H := M_i \oplus M'$. For convenience, we say that a server is *in* H if there is an edge in H incident on the server. Lemma 3, which says that any given server-vertex appears at most once in H, uses a "displacement sequence" in its proof which provides some intuition for the choice of H.

Lemma 2. *The servers used in M' are exactly the servers used in M_i.*

Proof. The name PERMUTATION in [4] comes from maintaining the invariant that "for all i, the vertices in S incident to an edge in M_i are exactly the vertices in S that are incident to an edge in P_i." By Lemma 3.2 of [4], S_i and S_{i-1} differ by exactly one server. Thus, by definition of how PERMUTATION chooses s_j^x, at each step i, M_i and $M_{i-1} \cup (r_i, s_j^x)$ have used the same servers.

Corollary 2. *H is a single alternating cycle.*

Proof. As in [4], this follows immediately from server vertices in M_i and M' being identical (Lemma 2).

Lemma 3. *If s_ℓ^1 is in H, then s_ℓ^2 is not in H, and if s_ℓ^2 is in H, then s_ℓ^1 is not in H.*

Proof. Suppose for the sake of a contradiction that H contains both the primary server and corresponding secondary server for some s_ℓ. By Lemma 2, s_ℓ^1 and s_ℓ^2 must each be used in both M_i and in M'. Let requests r_a and r_b be assigned to s_ℓ^1 and s_ℓ^2, respectively, by matching M_i. Let requests r'_a and r'_b be assigned to s_ℓ^1 and s_ℓ^2, respectively, in M'. To prove the lemma, it suffices to prove the following claim.

Claim: if $r'_a \neq r_a$ and $r'_a \neq r_b$, then $r'_b = r_a$ or $r'_b = r_b$. In other words, at least one of the two requests matched to a server of s_ℓ in M_i must also be matched to a server of

s_ℓ in M'. Assume not. So $r'_a \neq r_a$ and $r'_a \neq r_b$, and $r'_b \neq r_a$ and $r'_b \neq r_b$. Let s_j be the server-vertex assigned to r_i in M'.

Case $s_\ell \neq s_j$. Then, since $M' = M_{i-1} \cup (r_i, s_j)$, in M_{i-1} we must also have $r'_a \to s_\ell$ and $r'_b \to s_\ell$, where "$\to$" means "is assigned to." So upon the arrival of r_i, in the transition from M_{i-1} to M_i, both r'_a and r'_b were displaced by r_a and r_b.

Define the *displacement sequence of* r_i to be a sequence of server vertices and requests affected by the arrival of r_i, written as follows:

$$r_i \longrightarrow s_i \dashleftarrow r_1 \longrightarrow s_1 \dashleftarrow r_2 \longrightarrow s_2 ...$$

where forward-edges are from M_i and backward edges are from M_{i-1}. Here, r_1 is a request that was "displaced" from s_i upon the arrival of r_i; it was displaced to server-vertex s_1. Then r_2 is a request that was displaced from s_1 by r_1, and s_2 is the server-vertex it was displaced to, and so forth. Note that each server-vertex in this sequence can only have one incoming backward edge because it only has one incoming forward edge. Further note that if a server-vertex is not in the displacement sequence of r_i, then it must be matched to the same requests as it was in M_{i-1}, since otherwise the optimality of M_{i-1} or M_i or the assumption that M_i is the most similar optimal matching to M_{i-1} would be violated. So s_ℓ must be in the displacement sequence of r_i. Since s_ℓ has two displaced requests, r'_a and r'_b, then s_ℓ must appear twice in the sequence. But if it appears twice in the sequence, then there is a "cycle" in the sequence. Consider the displacements just in this cycle. The total cost of the forward edges in the cycle must be lower than the total cost of the backward edges, otherwise this cycle would not be present in the displacement sequence of r_i, it would just be cut out altogether (by optimality of M_i). But if the total cost of the forward edges is less than the backward edges, then M_{i-1} was not optimal.

Case $s_\ell = s_j$. Without loss of generality, let us assume that $r'_a = r_i$. Thus in M_{i-1}, only one request was assigned to s_ℓ and it was r'_b. So upon arrival of r_i, r_a was assigned to s_ℓ and r'_b was replaced by r_b. This means in the displacement sequence of r_i, s_ℓ again must appear twice, giving the same contradiction as in the previous case.

Now consider the server-vertices M_i uses exactly once (i.e. only their primary servers). The next lemma says at most one of these server-vertices can appear in H.

Lemma 4. *Let s_i^1 be in H. If an edge of M_i is not incident on s_i^2, then for all other servers s_j^1 in H, an edge of M_i must be incident on s_j^2.*

Proof. Note that by assumption, a secondary server cannot be used unless its primary server is used. Recall that $H := M_i \oplus M'$. Consider the arrival of r_i, and the change that occurs between M_{i-1} and M_i. If r_i is assigned by M_i to a server that was unused in M_{i-1}, then by definition of PERMUTATION, $s_j = s_i$, and hence $M' = M_i$. Thus H is empty, and no primary servers appear in H.

If in M_i request r_i is assigned to a server of vertex s_i that was used in M_{i-1}, the request assigned to that server in M_{i-1} must be reassigned, or displaced. Thus, we can construct a displacement sequence of r_i, denoted by a sequence of server-vertices and requests affected by the arrival of r_i, written as follows:

$$r_i \longrightarrow s_i \dashleftarrow r_1 \longrightarrow s_1 \dashleftarrow r_2 \longrightarrow s_2 \cdots s_t$$

where, again, forward edges are from M_i and backward edges are from M_{i-1}.

Note that by Lemma 2, the servers used in M_i are exactly those used in M'. Thus, since $M' = M_{i-1} \cup (r_i, s_j)$, s_j is used once more in M_i than it is in M_{i-1}, and all other server-vertices are used the same number of times in M_i as in M_{i-1}.

Only server-vertices that are in the displacement sequence can appear in H; all others have exactly the same requests assigned to them in M_i and M'.

Consider an arbitrary server s_ℓ in the displacement sequence that has its primary server but not its secondary server used in M_i. Look at the displacement sequence from s_ℓ to s_t. Since the secondary server at s_ℓ is unused in M_i, r_{k+1} was not forced to be displaced from s_k to s_{k+1}, but rather could have used said secondary server at s_ℓ. Thus, the forward edges (those from M_i) in the displacement sequence from s_ℓ to s_t must cost less than the backward edges (those from M_{i-1}). But this contradicts the optimality of M_{i-1}, since the assignments that represent these forward edges could have been made in M_{i-1} as well. Hence, there can be no edges from s_ℓ to s_t, and thus the only primary server in the displacement sequence that can be used without the corresponding secondary server being used is the final one, s_t.

Theorem 9. PERMUTATION *has a halfOPT-competitive ratio of $O(k)$ for the bottleneck matching problem.*

Proof. Let αk be the number of primary servers used by PERMUTATION. (This is the same as the number of primary servers used by M_k.) Since a secondary server is only used if its corresponding primary server is used, there are $(1-\alpha)k$ server-vertices with neither their primary nor secondary server used. Since exactly k requests are served, there must be $(1-\alpha)k$ secondary servers used. Together these guarantee $1 \geq \alpha \geq \frac{1}{2}$.

Let the bottleneck edge of the final PERMUTATION assignment be (r_i, s_j). Now consider the graph of H after the arrival of r_i. Recall that by Corollary 2, H is a single alternating cycle. As in [4], by the triangle inequality, the weight of the newest edge (r_i, s_j) is at most the aggregate weight of the edges in H minus its weight $d(r_i, s_j)$. Thus, if we can bound the number of edges in H by n, then the bottleneck edge for PERMUTATION is at most $n-1$ times the bottleneck edge in M_i, as the cost of the bottleneck edge only increases from M_{i-1} to M_i.

If for every primary server that is used in M_i, the corresponding secondary server is also used in M_i, i.e., $\alpha = \frac{1}{2}$, then by Lemmas 3 and 4, H is an alternating cycle with at most $k/2$ server vertices (and the same number of requests), for at most k edges. If instead the number of primary servers used exceeds the number of secondary servers used, then $\alpha k - (1-\alpha)k \geq 1$ which guarantees that $\alpha \geq \frac{k+1}{2k}$. By Lemmas 3 and 4, H contains at most $(1-\alpha)k+1$ servers, and thus the number of edges in H is maximized when α is as small as possible. Plugging in the lower bound on α gives $\frac{k+1}{2}$ servers, guaranteeing at most $k+1$ edges in H. Hence, in either case, PERMUTATION costs at most k more than the bottleneck edge in M_i; the optimality of M_k and the bottleneck edge of M_i monotonically non-decreasing as i increases complete the proof.

6 Conclusion

Resource augmentation results in a substantial improvement in the performance of the GREEDY algorithm for the bottleneck matching problem, from an exponential lower

bound to a guarantee linear in the number of requests. While still exponentially worse than its performance for the objective of minimizing total distance, it is a natural algorithm that is easy to implement. Two algorithms that perform notably better than GREEDY for the min-weight objective (PERMUTATION and BALANCE) also have linear competitive ratios for the bottleneck objective with resource augmentation. These results suggest that in some sense the bottleneck objective is more difficult than the total distance objective, as none of the three algorithms break the $\Omega(k)$ barrier for the bottleneck objective. Determining if the lower bound (under resource augmentation) is in fact $\Omega(k)$ remains an open question.

References

1. C. Chung, K. Pruhs, and P. Uthaisombut. The online transportation problem: On the exponential boost of one extra server. In *LATIN*, pages 228–239, 2008.
2. J. D. Hartline and T. Roughgarden. Simple versus optimal mechanisms. In *ACM Conference on Electronic Commerce*, pages 225–234, 2009.
3. R. Idury and A. Schaffer. A better lower bound for on-line bottleneck matching, manuscript. 1992.
4. B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993. Preliminary version appeared in *SODA*, pp. 231-240, 1991.
5. B. Kalyanasundaram and K. Pruhs. The online transportation problem. *SIAM J. Discrete Math.*, 13(3):370–383, 2000. Preliminary version appeared in *ESA*, pp. 484-493, 1995.
6. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47:617–643, July 2000. Preliminary version appeared in *FOCS*, pp. 214-221, 1995.
7. S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127:255–267, May 1994.
8. C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002. Preliminary version appeared in *STOC*, pp. 140-149, 1997.
9. T. Roughgarden and É. Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, 2002. Preliminary version appeared in *FOCS*, pp. 93-102, 2000.