Greedy Is Good: An Empirical Evaluation of Three Algorithms for Online Bottleneck Matching

Barbara M. Anthony, Christine Harbour, and Jordan King

Mathematics and Computer Science Department,
Southwestern University, Georgetown, Texas, USA
anthonyb@southwestern.edu,
{harbourc, king3}@alumni.southwestern.edu

Abstract. We empirically evaluate the performance of three approximation algorithms for the online bottleneck matching problem. In this matching problem, 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 assigned server. We consider the naïve GREEDY algorithm, as well as PERMUTATION, and BALANCE, each of which were constructed to counter certain challenges in the online problem. We analyze the performance of each algorithm on a variety of data sets, considering each both in the original model, where applicable, and in the resource augmentation setting when an extra server is introduced at each server-vertex. While no algorithm strictly dominates, GREEDY frequently performs the best, and thus is recommended due to its simplicity.

1 Preliminaries

Fundamentally, matching problems are often quite simple to state: given two sets, match each item in the first set to a distinct item in the second set. Yet matching problems are broadly applicable and sometimes deceptively complex. Approximation algorithms offer a pragmatic way to achieve an acceptable solution to NP-complete problems and have been extensively studied and developed in recent decades. Approximation algorithms are measured by their theoretical guarantee, quantifying the worst possible performance. Yet, these worst-case scenarios may not be frequently realized in practice, and thus other metrics may be more appropriate in characterizing algorithms. Likewise, one approximation algorithm may have a worse theoretical guarantee than another, but perform better in typical situations. Hence, the empirical evaluation of approximation algorithms provides additional insight into the strengths, weaknesses, and applicability of various algorithms.

We consider several algorithms for the online bottleneck matching problem. Informally, in the online bottleneck matching problem, a set of servers are known

from the beginning, and a set of requests arrive over time. As the requests arrive, each must be matched to a server, and the quality of the solution is measured based on the length of the longest edge in the matching. The performance of the approximation algorithm is compared to the *bottleneck cost*, the cost of the longest edge in the matching, of the optimal offline solution, where all requests and servers are known in advance.

In Section 1, we formally define the problem, discuss recent work on the problem, describe the algorithms studied, and consider other empirical evaluations. Section 2 details the implementation of the algorithms and a strategic brute force solution. Section 3 describes some of the data sets considered, and the analysis thereof is provided in Section 4, before concluding in Section 5.

1.1 Online Bottleneck Matching

Formally, in the online bottleneck matching problem, we are given a set of server-vertices $S = \{s_1, s_2, \ldots, s_k\}$ that lie in a metric space. Over time the requests arrive at request-vertices $R = \{r_1, r_2, \ldots, r_k\}$. The online algorithm only learns the value of the next request-vertex in the sequence after it has assigned the previous request. Thus, upon the arrival of request vertex r_i, the online algorithm must permanently and irrevocably assign an unused server-vertex s_j to service that request. The cost of the assignment is the distance between these two vertices, and the bottleneck cost (or distance) of the overall assignment is the cost of the most expensive assignment made. The online algorithm attempts to minimize the bottleneck cost of the overall assignment.

As is standard with online algorithms, the performance of an algorithm is evaluated using competitive analysis. Let OPT denote an assignment which achieves the optimal offline bottleneck cost. We will also use OPT to refer to said optimal offline bottleneck cost. An online algorithm is thus deemed α-competitive if the worst-case ratio of the online bottleneck cost to the bottleneck cost of OPT is at most α for all possible instances. When looking at a particular instance of a problem, we consider the ratio of the online bottleneck cost on that instance to the bottleneck cost of the optimal solution for that instance.

The three algorithms we consider were studied in [2] for the online bottleneck matching problem, and were previously introduced in the context of total-weight matching [9, 11, 10], where the goal is to find the matching with least total cost, rather than least bottleneck cost. Observe that said problem with the *min-weight* objective is equivalent to minimizing the average distance between request-vertices and server-vertices. The performance of GREEDY was shown to be exponential in [2], which also confirmed the prior observation of [9] that PERMUTATION was linear. Using an instance where all the servers and requests lie on a single line, Idury and Schäffer [8] proved that no algorithm can achieve a competitive ratio better than approximately $1.5k$.

This 'negative' lower bound result motivated the study in [2] using a weak-adversary model, namely resource augmentation. In this setting, they consider how well the online algorithm does when it has two servers available per server-vertex, while leaving the optimal offline solution unchanged. This type of bicriteria result has also been used in the min-weight matching problem, with [10] showing that GREEDY does substantially better with resource augmentation for the total-weight objective. Yet GREEDY still had its limitations, leading [10] to propose the algorithm BALANCE, a modification of GREEDY which uses a penalty cost to decide when to use the second server at a given server-vertex.

Following the notation of [10, 2] we use the term *halfOPT-competitive* ratio when referring 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. In [2] it was shown that for the bottleneck objective all three algorithms have a halfOPT-competitive ratio of at least $\Omega(k)$, and that GREEDY and BALANCE have upper bounds of k in that setting, while PERMUTATION has an upper bound of $k-1$. Given the comparable performance of the three algorithms, they support the use of GREEDY for the bottleneck objective with resource augmentation since it is the simplest of the three algorithms.

The online bottleneck matching problem has also been studied in the context of a Serve-or-Skip bicriteria analysis model, where the online algorithm may reject or skip up to a specified number of requests [3]. That model is not studied in this work, but remains an interesting direction for future work.

1.2 Three Approximation Algorithms

We briefly describe the three approximation algorithms for completeness. Note that in the resource augmentation model, we may denote the original (primary) server with a superscript of 1, and the additional (secondary) server with a superscript of 2. While this is for notational convenience, without loss of generality all algorithms will choose the smallest available superscript for a given server.

GREEDY behaves in the natural way, by greedily selecting the closest available server each time a request is revealed, with ties broken arbitrarily. BALANCE, proposed in [10], again follows the greedy paradigm, but only applies in the resource augmentation setting. Informally, the algorithm pays a multiplicative distance penalty to assign a request to the second server at a server location. More precisely, BALANCE determines *pseudo-distances* from a request r_i, where the pseudo-distance from r_i to s_j^1 is $d(r_i, s_j)$, the actual distance in the metric, while the pseudo-distance from r_i to s_j^2, the secondary server at the same location, is $c \cdot d(r_i, s_j)$ for a constant c. BALANCE then assigns requests to servers greedily using the pseudo-distances, though the cost paid remains the actual distance in the metric. (Thus, note that if $c = 1$, BALANCE is in fact GREEDY in the resource augmentation model.) In [10], the constant for BALANCE was specified as $c > 5 + 4\sqrt{2}$.

The final algorithm we consider, PERMUTATION, is given in [9], with similarities to an algorithm studied by [11]. In essence, when each request arrives, the algorithm computes the optimal offline solution based on the requests thus far, and assigns the new request to the server used in this optimal offline solution which is not used in the current online solution. Formalizing this idea, as in [9], if $R_i \subset R$ are the first i requests, a *partial matching* of R_i is a perfect matching of R_i with a subset of the servers of S. Define M_i to be the edges that form a minimal weight partial matching on R_i which minimizes $|M_i - M_{i-1}|$. Let $S_i \subset S$ be the servers used in M_i. PERMUTATION constructs its assignment for $i+1$ requests by computing M_{i+1}, maintaining the existing assignments from the first i requests, and assigning request r_{i+1} to the unique server $s \in S_{i+1} - S_i$. Recall that the offline solution only has one server per server-vertex, even in the resource augmentation setting.

1.3 Empirical Evaluation of Approximation Algorithms

While much effort has been put into developing approximation algorithms for NP-complete problems, the theoretical guarantee is, of course, a measure of the worst possible performance. In many cases, worst-case analysis is performed rather than average-case analysis, in part because determining an appropriate set of inputs to consider for average-case analysis is challenging. Yet, for many practical problems, there is interest in typical behavior. Thus, there is a growing body of work which empirically evaluates the performance of various approximation algorithms, not only for network-based problems that are most similar to what we consider here, but also in domains such as probabilistic encoding [13]. Even for problems with constant approximation guarantees, like the well-studied k-center problem with well-known 2-approximation algorithms, [7] shows that increasing or decreasing the value of k by 1 can in fact result in the actual performance of the approximation algorithm moving from near one extreme to the other. In [12, 6] algorithms and heuristics are both evaluated for the single-source unsplittable flow problem. They seek to find not only how close to the fractional optimum the calculated solutions are, but also how the solutions compare on running time, concluding that the actual behavior is often much better than the theoretical guarantee.

Finally, we are certainly not the first to conclude that greedy approximation algorithms do well. We note several papers that also use a variant of the expression "greedy is good" in their titles, including [14, 1, 15] considering the sparse approximation problem over redundant dictionaries, server placement in sensor networks, and multi-constrained quality-of-service routing, respectively.

2 Implementation Decisions

Each of the algorithms was implemented in C++. While the code was designed to be reasonably efficient, the implementations were not optimized for speed, an issue previously addressed in [6], as the focus of this work is comparing the performance guarantee of the objective function. Instances with k in the hundreds can easily run in minutes on a standard laptop for all of the approximation algorithms, in both the original and resource augmentation models. However, the brute force algorithms for OPT and halfOPT are, not surprisingly, much slower, with an instance for halfOPT with $k = 13$ needing five hours, and larger instances requiring days. Again, while some improvements in this area are possible, it is not the focus of this work.

Two tie-breaking conditions are implemented without loss of generality. By default when in the resource augmentation model, all implementations choose the server labeled s_i^1 before choosing s_i^2. A small $\epsilon > 0$ is added to the distance to each s_i^2 at runtime in order to enforce the decision that s_i^1 should be chosen first as the lower-cost server. In the case of all other ties, the server with the smallest label is chosen; thus if servers s_i and s_j are equidistant, and $i < j$, s_i is chosen.

2.1 The Data Files

Assignment decisions are based upon the distance between a given server and a given request. Thus for each data set there is a file containing a two-dimensional distance matrix which allows for efficient access to these distance values. Formally, the ijth entry in the distance matrix is the distance from request r_i to server s_j. Naturally, there are additional data files for each considered permutation. Note that the input file does not need to be modified when resource augmentation is used. As described above, a tie-breaking ϵ is used in the implementation itself, and in cases where pseudo-distances are used, the algorithm makes those calculations at runtime rather than modifying the data file. Additionally, the algorithms are implemented to consider only one row, or one request, of the distance matrix at a time. This preserves the online nature of the problem.

2.2 Greedy Algorithms

The implementation of GREEDY, the natural greedy strategy, was straightforward. Costs are read into a vector one request at a time. Each request is then greedily assigned to the closest available server using a boolean value to determine if a server is available. In the resource augmentation condition, the number of considered servers is doubled and the perturbation of ϵ is added to the cost of each s_i^2 at runtime. Requests are then greedily assigned, in both the original model and the resource augmentation setting.

Since the BALANCE algorithm is in fact a greedy algorithm with specialized distances, the implementation of BALANCE was quite similar to that of GREEDY with resource augmentation, the only setting in which BALANCE is defined. While resource augmentation is a theoretically powerful idea, in practice its implementation is straightforward because the algorithm simply needs to double the available number of servers. With the tie-breaking conventions previously described, it greedily assigns the current request to the closest available server, where closest is defined appropriately for the algorithm.

As noted, in BALANCE, the algorithm considers a pseudo-distance when choosing to assign a request to a secondary server (a multiplicative factor of c) but that penalty is not considered when calculating the bottleneck cost. While the $c > 5 + 4\sqrt{2}$ specified in [10] arises in the theoretical proof of its performance guarantee, the algorithm does not require any specific value of c, recalling that $c = 1$ is precisely GREEDY. We thus choose values of c both based on that theoretical value, and based on the distances in our data set. Intuitively, if a secondary server is chosen only when it is worthwhile to pay an extra factor of c, the performance of the algorithm can depend greatly on the choice of c. BALANCE is likely to perform well with what might be called judicious use of secondary servers, that is, using them when there are noticeable benefits, but not at the risk of having them unavailable later when the benefit would be even greater. For that reason, we often implement BALANCE with multiple c values.

2.3 PERMUTATION

PERMUTATION is defined for both the original model and resource augmentation, with many similarities between the two implementations. While the behavior of PERMUTATION is fairly straightforward to describe at a high level, its implementation is more involved because of the need to solve a partial matching as a subroutine. For partial matching, we employed the LEMON (Library for Efficient Modeling and Optimization in Networks) C++ template library [5], specifically the Max-Weight Matching algorithm and the directed graph data structure. (Since our problem is in terms of minimum costs, we were required to translate our values to a max-weight scenario by subtracting each value from a sufficiently large constant.)

Once the appropriate conversions have been made, the LEMON graph is initially constructed with each server represented by a node. A set is also created to hold the servers that have been assigned to a request. In each iteration, a new node r_i is constructed for the incoming request and edges with appropriate weights are created based on the distance values in the max-weight distance matrix. The max-weight matching algorithm is then run, which finds an optimal matching given the servers and subset of requests that have arrived so far. The servers in the new matching are compared to a set of servers that have been used in previous iterations. The server in the current matching that is not a member of the set is then

assigned to service request r_i and is added to the set of matched servers. In the LEMON graph structure, nodes are indexed from 0 as they are added. In the resource augmentation condition, the initial servers are indexed $0, \ldots, k-1$, and the additional servers at each location are $k, \ldots, 2k-1$, ensuring the indexes of s_i^1 and s_i^2 are an additive k apart. When the arcs to additional servers are added, the aforementioned ϵ must be subtracted from the weight to enforce that in the max-weight scenario the initial server will be chosen first.

2.4 OPT

To compute the optimal solution (on small instances, since naturally the problem quickly becomes intractable) to the offline bottleneck matching problem, it must be calculated by brute force. The natural approach is to fix an ordering of the servers, and try all possible $k!$ permutations of the k requests, determine the bottleneck cost of each such matching, and take the smallest. We improve slightly upon that naïve strategy by observing that since we seek the minimum bottleneck cost, we can stop computing the cost of any matching where the current bottleneck edge is more expensive than that of the best matching found thus far. Likewise, if we have already run one of our approximation algorithms on a given instance, since the optimal solution must be at least as good, we can provide the bottleneck cost of the approximation algorithm as the starting minimum bottleneck cost. While these measures provide some time savings, they do not fundamentally change the factorial runtime generally necessary for computing an optimal solution.

In the special case where all the servers and requests lie on the line, the optimal solution is in fact easy to compute offline. It can readily be observed that if the requests and servers are each sorted from left to right on the line, the leftmost request is assigned to the leftmost server, and so forth. Thus, the runtime is dominated by the $k \log k$ time required to sort each of the sets. Note, however, that such an assignment does not extend to the resource augmentation model; requests will now be assigned left to right to a *subset* of size k, giving an exponential runtime. Observe that in this special case, the original data file with locations of the servers and requests is more useful than the two-dimensional distance matrix used elsewhere.

3 Data Sets Analyzed

In empirical evaluation, the choice of data sets analyzed naturally impacts the results. It is of course natural to consider the instances which provided the matching bounds for the theoretical performance guarantees of some of the algorithms, including those studied in [2]. Note that while all of the data sets in [2] involve metrics, not all of them are Euclidean, validating the decision to have data input files maintain the distance between every request and server. Random data sets are

of course useful, especially because they can avoid some of the potential biases of constructed data sets. For random data sets, we fixed a number k, and independently and randomly generated k server locations and k request locations in a 100 by 100 region of the Euclidean plane. Observe that since distances and locations are recorded as real numbers, the precise location in the plane is not important, and the size is also without loss of generality due to scaling.

The line, though simple to visualize, proves worth of study in this problem. Though the optimal offline solution is straightforward, in the online scenario it is much more challenging, as evidenced by it forming the basis of the general lower bound for any algorithm for the online bottleneck matching problem [8]. Only recently was the first $o(n)$ algorithm for online matching on a line (with the total-weight objective, not the bottleneck objective) given [4]. Thus, we looked at randomly constructed instances on the line. For each data set, we randomly generated k server locations along the line, and then randomly generated k request locations along the line, and then picked some of the $k!$ permutations of the possible arrivals of the requests. (Note that it is unnecessary to permute the ordering of the server vertices, as all of them are known prior to the first request arriving.)

As one example of a real-world data set, we consider the problem of sending ambassadors from CS clubs at each of the fifty largest colleges and universities in the United States to the fifty largest population centers in the United States to provide CS outreach to school districts. The requests are thus the fifty largest colleges and servers are the fifty largest population centers in the United States and the objective is a matching which minimizes the farthest any college's CS club must travel.[1] While these sets and rankings may change over time or be disputed, the actual choices of values are provided for illustration purposes, and could easily be adjusted. Ordering the requests by college size order (increasing or decreasing) are only two of many possible request arrival orders of interest. There are also multiple potential measures of distance. For simplicity, we chose to use the distance as the crow flies, though mapping software could be used to compute driving times, or airline routes could be used to determine expected travel time. Thus, our distances are measured in terms of distances between coordinates in Euclidean space when the cities and colleges were plotted; they are not in miles, but could be scaled accordingly.

4 Results and Discussion

We cannot definitively conclude that one of the algorithms is always best, but our empirical evaluation does confirm that the theoretical guarantees and their relative values are not generally indicative on the algorithms' performance on particular

[1] The population centers were obtained from http://www.infoplease.com/ipa/a0763098.html, and the colleges from http://www.matchcollege.com/top-colleges, excluding online colleges, both as of April 2015.

instances. Thus, while GREEDY has the worst theoretical guarantee of the studied algorithms, it frequently has the best performance, and has the added advantage of being straightforward to explain and implement. Additionally, while the algorithms may do far better than their theoretical guarantee suggests, in the random settings and those derived from real-world data, we also find that for sufficiently large instances, the approximation algorithms never achieve the optimal value, in either the standard model or resource augmentation. Yet limited conclusions can be drawn, even about the performance of algorithms on particular types of scenarios, since on the same exact sets of servers and requests, with a different ordering of the request arrivals, GREEDY can sometimes be better than PERMUTATION, and sometimes worse, in meaningful ways.

Because BALANCE is a greedy algorithm, it is natural to try to make direct comparisons between it and GREEDY. By construction, BALANCE in the resource augmentation setting, the only setting where it is defined, must do at least as well as GREEDY on the same instance in the standard model. However, there is no corresponding claim about BALANCE as compared to GREEDY in the resource augmentation setting. As noted in the following portions, BALANCE can actually do notably worse than GREEDY in the resource augmentation setting since in some situations the penalty factor c can cause it to not pick a server that would otherwise be a good choice. This is more pronounced when c is large compared to the maximum distance from a server to a request. Thus, if BALANCE is chosen as the approximation algorithm, a user should carefully consider the choice of c, likely test multiple c values, and note that a slight change in that penalty can significantly change the resulting bottleneck cost.

As mentioned, it can sometimes be feasible to compute OPT but halfOPT is computationally intractable. Naturally, the bottleneck cost in the standard model is always an upper bound on the bottleneck cost in the resource augmentation model. In some cases, they can in fact be identical – consider a scenario where the servers are at $\{2, 4, 6, \ldots, 2k\}$ on the line and the requests arrive in order at $\{1, 3, 5, \ldots, 2k-1\}$, where the optimal bottleneck cost is 1 in either model; tie-breaking rules or minor distance perturbations can easily be employed so that GREEDY and the other approximation algorithms will obtain the optimal solutions. However, the bottleneck costs in the two models are frequently quite different, with halfOPT often significantly smaller than OPT, which is again not surprising, and a frequent motivation for resource augmentation models, since the additional resources can sometimes dramatically improve the solution. In the remainder of this section we provide specific instances from the data sets previously described.

4.1 Examples Illustrating Theoretical Bounds

The instances of the examples given in [2] that provide the basis for the theoretical bounds on the algorithms primarily served as one of many checks of the validity

of our implementations. As expected, the empirical performance matched the theoretical bounds for the specified algorithms, and since the instances were typically developed to highlight a limitation of a particular algorithm, often did markedly better for the other algorithms. For example, for all values of k tested, the instance described in Theorem 1 of [2], where requests and all but one server are placed at one less than successive powers of 2 showed that GREEDY does exponentially poorly on this instance, while PERMUTATION is an $O(1)$-approximation. Thus, as the previous work indicated, these theoretical performance guarantees are in fact realizable with particular instances. However, as the rest of our instances show, the approximation algorithms often perform markedly better than their theoretical guarantees, which should thus not be assumed to represent the expected or typical performance.

4.2 Random, Constrained to a Line

We next consider the instances where k requests and k servers are randomly chosen from an interval on a line. Recall that as detailed in Section 2.4, the optimal solution can be efficiently computed for large values of k in the standard model, but not in the resource augmentation model. Thus, for the standard model, Table 1 provides details on both approximation algorithms as well as the optimal solution, but for the resource augmentation model reports only on the approximation algorithms. It provides results for two values of c for BALANCE, both the $c \approx 10.66$ that is used in the theoretical guarantee, and a much better penalty factor of $c = 2$ given the points all lie within an interval of size 100. The data were generated as described in Section 3, and the table reports on an arbitrary permutation of the requests for each of the generated datasets.

Table 1. The bottleneck cost for seven instances, each with k randomly generated servers and k randomly generated requests which all lie on a line of length 100, under the different approximation algorithms for both the standard model and resource augmentation, as well as the bottleneck cost of the optimal solution in the standard model.

		$k = 50$	$k = 75$	$k = 100$	$k = 125$	$k = 150$	$k = 175$	$k = 200$
Standard Model	GREEDY	70.3113	59.0686	85.1544	85.4344	48.8434	66.4495	57.8651
	PERMUTATION	70.3113	59.0686	85.1544	85.4344	48.8434	66.4495	57.8651
	OPT	12.0686	8.4615	9.41443	14.4142	10.6674	10.1597	5.19417
Resource Augmentation	GREEDY	10.0154	10.9578	2.81056	4.5045	4.23073	4.77434	2.23169
	PERMUTATION	12.867	10.958	5.0171	3.9151	5.653	4.7741	2.2321
	BALANCE $c \approx 10.66$	23.6172	17.8645	9.25468	13.9027	7.20631	10.3744	6.91647
	BALANCE $c = 2$	8.98231	11.0298	3.80985	3.83119	2.89899	6.07102	2.8076

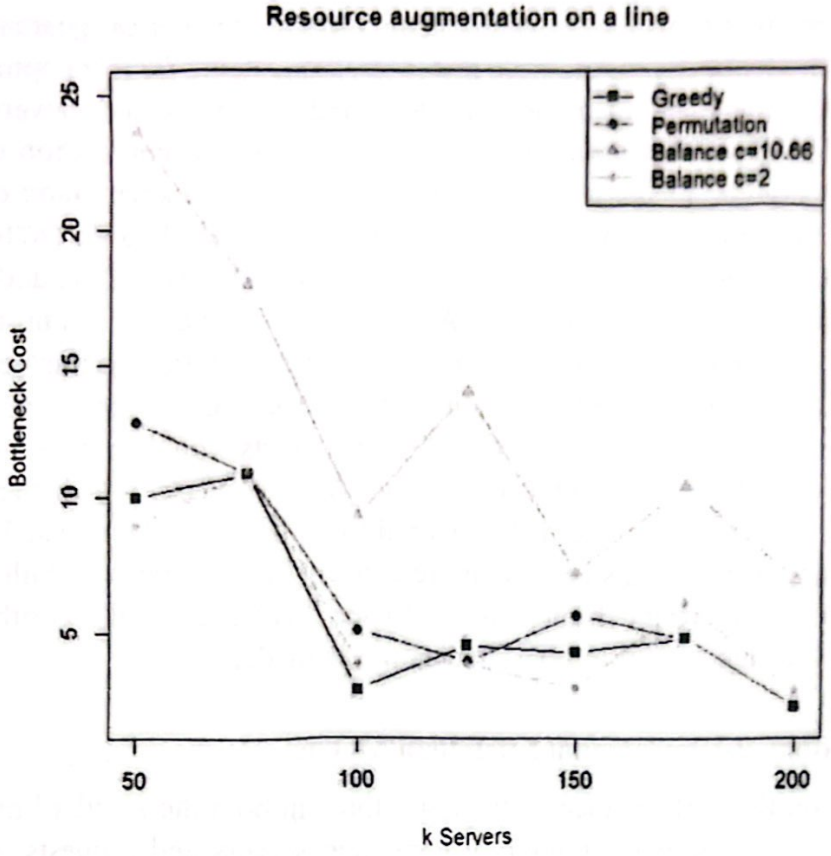

Fig. 1. Performance of the three approximation algorithms, with two c values for BALANCE, in the resource augmentation model for seven instances, each with k randomly generated servers and k randomly generated requests which all lie on a line of length 100.

Observe that on all instances, GREEDY and PERMUTATION performed identically in the standard model, yet were noticeably different from OPT. Because the interval in which the points lie is bounded, this size (100) naturally provides an upper bound on the performance of any of the algorithms which is far more restrictive than the theoretical bound for either algorithm (exponential for GREEDY, linear for PERMUTATION [2]). Thus, in that regard, the performance of the approximation algorithms in the standard model can be considered quite poor, since all but one has a bottleneck cost that is at least half of the total distance of the region. Since the density of points within the region increases as k increases, it is unsurprising that the smallest OPT occurs with the largest k, yet it is also logical that the online algorithms may in fact encounter certain challenges with more dense points, and do not necessarily trend comparable to OPT. Hence, these instances reinforce the notion that the line, simple as it may be, remains a challenge for the bottleneck matching problem. Moreover, they highlight that while two ap-

proximation algorithms may have markedly different theoretical guarantees, they can perform identically on multiple instances, while being far from optimal.

Unlike in the standard model, GREEDY and PERMUTATION never have the same bottleneck cost in these instances in the resource augmentation model. In fact, as illustrated in Figure 1 which plots the resource augmentation data from Table 1, while GREEDY has a smaller bottleneck cost than PERMUTATION much of the time, in one instance PERMUTATION outperforms GREEDY, and in some the difference in costs is quite small. Again, computing halfOPT is prohibitively expensive. All three approximation algorithms have theoretical guarantees that are linear in k as shown in [2], though their performance is also again bounded by the size of the line segment, 100, yet neither of those facts would be observable from this data alone. While none of the algorithms give bottleneck costs that are strictly decreasing as k increases, that is the overall trend for all of them, which is again not surprising as the points become more dense within the region. With resource augmentation, we are much more likely to see this trend as all algorithms have some means of recourse if a 'poor' decision was made.

4.3 Random, 2-Dimensional Euclidean Metric

We report on the performance of the algorithms in both the standard model and resource augmentation for randomly generated servers and requests in a two-dimensional 100×100 region where distances obey the Euclidean metric, for two values of k, namely $k = 13$ and $k = 200$. In each case, we consider five arrival orders of the requests, chosen arbitrarily. For $k = 13$ we plot competitive ratios representing the ratio of the cost of the algorithm's solution to the cost of the optimal solution on that given instance in Figure 2, and precise numerical values of the calculated and optimal bottleneck costs are reported in Table 2.

Table 2. Performance of the different algorithms on $k = 13$ random requests and servers in two dimensions, with five different arrival orders of the requests.

		Order 1	Order 2	Order 3	Order 4	Order 5
Standard Model	GREEDY	94.9626	78.6556	79.71	72.1809	61.7316
	PERMUTATION	105.798	99.297	90.787	99.075	76.635
	OPT	48.0173				
Resource Augmentation	GREEDY	57.8078	57.8078	45.6416	45.6416	45.6416
	PERMUTATION	99.075	99.075	45.6421	59.1161	99.075
	BALANCE $c \approx 10.66$	94.9626	78.6556	79.71	45.6416	61.7316
	BALANCE $c = 2$	72.1809	61.7316	57.8078	57.8078	61.7316
	halfOPT	45.6416				

Not surprisingly, resource augmentation appears to be less useful for randomly selected points in two dimensions than in one dimension, as evidenced by the

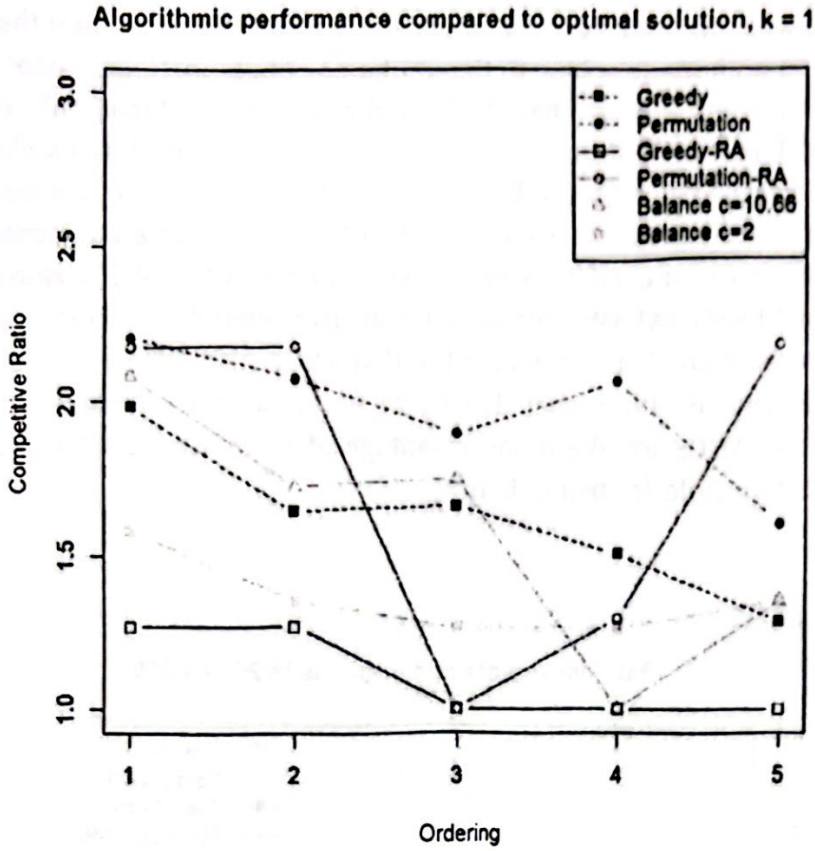

Fig. 2. Ratio of the bottleneck cost of the approximation algorithm to the optimal solution for each of the request arrival orderings on $k = 13$ randomly generated requests and servers in a 100×100 region with Euclidean distances.

values for OPT and halfOPT. Yet the way that the approximation algorithms make their choices show that the availability of a second server to perhaps rectify poor prior choices can be quite advantageous for the tested algorithms. Observe that GREEDY was the best overall performer in both models, though in one instance BALANCE also obtained the optimal solution, notably with $c \approx 10.66$ but not with $c = 2$ for that same instance. Yet in the other four instances, BALANCE with $c = 2$ was at least as good as BALANCE with ≈ 10.66, sometimes significantly so. Again, the ordering of the requests makes a substantial difference; PERMUTATION is one of the worst performers in terms of the competitive ratio in both models, yet it does near-optimally for the third ordering in the resource augmentation setting. As before, the bounded size of the region ensures that all algorithms perform markedly better than the theoretical guarantee.

In additional instances with k between 10 and 15 whose results are not explicitly provided here, GREEDY was again better than or at least as good as PERMUTATION 90 percent of the time in the standard model, and was the best of the

tested algorithms more than two-thirds of the time in the resource augmentation model. Likewise, in instances with larger k, GREEDY was again best the majority of the time in both models. Due to the limitations of computing optimal solutions for large k (in both the original model and resource augmentation), we cannot compute OPT or halfOPT for $k = 200$, but plot the bottleneck costs obtained by the approximation algorithms in Figure 3. Note that while the bottleneck costs for GREEDY seem to have typically been cut in half by resource augmentation, we cannot assume that same relationship between OPT and halfOPT. Again GREEDY outperforms PERMUTATION, this time for all five orderings in both models. For these instances, a smaller constant c for BALANCE also appears to be advantageous, likely because the higher density of points with a comparatively large k allows for BALANCE to take more advantage of secondary servers, particularly with a lower threshold for using them.

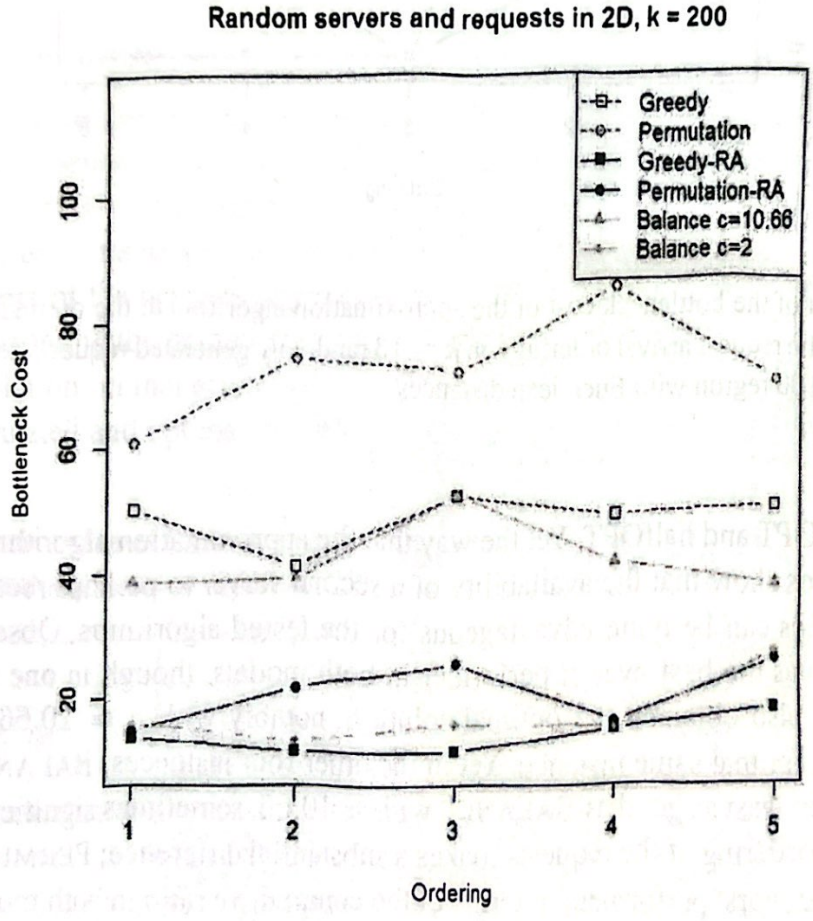

Fig. 3. Bottleneck costs obtained by the approximation algorithms for each of the request arrival orderings on $k = 200$ randomly generated requests and servers in a 100×100 region with Euclidean distances.

4.4 Colleges and Population Centers

Finally, we consider the matching of colleges to population centers with $k = 50$. Note that it was in fact possible to compute OPT and halfOPT in a matter of minutes, even though for similar size data sets with random data in two dimensions the computations were prohibitively expensive. This difference is likely due to the structure that is inherent in this real set, as well as the time-saving measures described for implementing the brute force solution which discard potential matchings that are more expensive than the best previously discovered.

Table 3. Performance of the various algorithms (and the optimal solution) in the original model and with resource augmentation on a colleges and population centers data set ($k = 50$), when the requests (colleges) arrive in order of increasing size, decreasing size, and alphabetically.

		Increasing Size	Decreasing Size	Alphabetical
Standard Model	GREEDY	36.8705	39.2734	36.6194
	PERMUTATION	37.8700	39.2734	36.183
	OPT	30.0832		
Resource Augmentation	GREEDY	12.2332	11.3464	11.447
	PERMUTATION	13.2331	11.9905	13.4081
	BALANCE, $c \approx 10.66$	36.4335	38.1106	34.6606
	BALANCE, $c = 2$	18.8048	15.1327	16.9319
	halfOPT	9.8005		

Some numerical results are highlighted in Table 3. While none of the algorithms achieve the optimal solution in either the original model or under resource augmentation, GREEDY typically (but not always) provides the best results for both settings. We do not claim, however, that this will always be the case for any data set; in fact it is possible that some ordering of these exact colleges and population centers could have another algorithm outperform GREEDY for resource augmentation, just as PERMUTATION is slightly better than GREEDY when the colleges are ordered alphabetically in the standard model. Observe that GREEDY and PERMUTATION have similar or identical bottleneck costs in the standard model, and that these values are substantially closer to OPT than they were on the line. Again, the bounded region ensures a maximum bottleneck cost, which in this case is 42.9691. Thus, while the ratio of the performance of the approximation algorithms to the optimal solution in the standard model is fairly low, it is in fact not necessarily that far from the bottleneck cost of the worst possible matching.

Observe that in the standard model with three plausible orderings of interest in the real world, GREEDY outperforms PERMUTATION for one ordering, ties it for another, and underperforms it in a third, though the difference is smaller

when GREEDY underperforms than outperforms. Yet in the resource augmentation model, GREEDY outperforms all of the tested algorithms on the three orderings. Again, due to the nature of the bounded region and its size, BALANCE does much better with $c = 2$ than with $c \approx 10.66$, and GREEDY, which is BALANCE with $c = 1$ does the best. While the alphabetical ordering produced the worst results for the standard model, it was the ordering in which the algorithms generally performed the best for resource augmentation.

5 Conclusions

Approximation algorithms are increasingly important for solving intractable problems, yet an algorithm is evaluated based on competitive analysis of the worst-case online performance of an instance compared to the optimal offline solution. We continue the trend of some recent work in showing that empirical evaluation of approximation algorithms adds a valuable dimension to their study. Moreover, we show that the relative upper bounds on the theoretical performance of approximation algorithms are not necessarily good indicators of the algorithms' performance on typical instances. In particular, in the online bottleneck matching problem, we show that the 'obvious' greedy algorithm is frequently the best of the studied approximation algorithms for that problem in the standard model, despite having a significantly worse theoretical guarantee. Likewise, though the algorithms considered all have a linear guarantee in the resource augmentation model, their actual performance can vary greatly, with GREEDY again often the best, sometimes obtaining the optimal solution.

Furthermore, knowledge about the structure of the data set can guide the choice of algorithm. We observe different performance for two-dimensional random data than one-dimensional, and if the data lies in a known bounded region, that can be useful in determining the choice of constant c used in BALANCE. GREEDY is the easiest to implement, and frequently the best, leading to its recommendation as the default algorithm if only one can be implemented. Yet, all of the approximation algorithms ran in minutes, and thus there are scenarios where calculating the results of multiple approximation algorithms is appropriate.

Future work includes evaluating approximation algorithms on different models for this problem, including the Serve-or-Skip model introduced in [3] and exhaustively testing all $k!$ orderings for some instances to better quantify the percentage of time that GREEDY is best.

Acknowledgments

This research was partially supported by the Collaborative Research Experience for Undergraduates program of the Computing Research Association Committee on the Status of Women in Computing Research in conjunction with the Coalition to Diversify Computing, funded by the National Science Foundation.

References

1. Z. Abrams and J. Liu. Greedy is good: On service tree placement for in-network stream processing. In *26th IEEE International Conference on Distributed Computing Systems, ICDCS 2006*, page 72, 2006.
2. B. M. Anthony and C. Chung. Online bottleneck matching. *J. Comb. Optim.*, 27(1):100–114, 2014. Preliminary version appeared in *COCOA*, pp. 257-268, 2012.
3. B. M. Anthony and C. Chung. The power of rejection in online bottleneck matching. In Z. Zhang, L. Wu, W. Xu, and D.-Z. Du, editors, *Combinatorial Optimization and Applications*, LNCS, pages 395–411. Springer International Publishing, 2014.
4. A. Antoniadis, N. Barcelo, M. Nugent, K. Pruhs, and M. Scquizzato. A $o(n)$-competitive deterministic algorithm for online matching on a line. In E. Bampis and O. Svensson, editors, *Approximation and Online Algorithms - 12th International Workshop, WAOA 2014*, volume 8952 of *LNCS*, pages 11–22. Springer, 2014.
5. B. Dezs, A. Jüttner, and P. Kovács. LEMON - an open source C++ graph template library. *Electron. Notes Theor. Comput. Sci.*, 264(5):23–45, July 2011.
6. J. Du and S. G. Kolliopoulos. Implementing approximation algorithms for the single-source unsplittable flow problem. *J. Exp. Algorithmics*, 10, Dec. 2005.
7. C. Harbour and J. King. An empirical evaluation of a k-center 2-approximation algorithm. In *Student Paper E-Journal, The Consortium for Computing Sciences in Colleges: South Central Region*, 2015.
8. R. Idury and A. Schaffer. A better lower bound for on-line bottleneck matching, manuscript. 1992.
9. B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993. Preliminary version appeared in *SODA*, pp. 231-240, 1991.
10. 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.
11. 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.
12. S. G. Kolliopoulos and C. Stein. Experimental evaluation of approximation algorithms for single-source unsplittable flow. In *Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 328–344. Springer-Verlag, 1999.
13. I. Rish, K. Kask, and R. Dechter. Empirical evaluation of approximation algorithms for probabilistic decoding. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 455–463. Morgan Kaufmann Publishers Inc., 1998.
14. J. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, Oct 2004.
15. G. Xue and W. Zhang. Multiconstrained QoS routing: Greedy is good. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 1866–1871, Nov 2007.