Empirical Competitive Analysis for the Online Assignment Problem with ML Predictions

Clarence Gabriel R. Kasilag, Pollux M. Rey, and Jhoirene B. Clemente

Department of Computer Science
University of the Philippines Diliman
{crkasilag,pmrey,jbclemente}@up.edu.ph

Abstract. The online assignment problem, also known as the online-weighted bipartite matching, produces the smallest weight perfect matching given a complete bipartite graph. The problem is a variant where one part of the graph is known in advance, while the other part is revealed one vertex at a time. Moreover, all the incident edges are revealed as a new vertex arrives. This computational problem plays an important role in the fields of operational research and computer science. Due to the incomplete information about the input, it is difficult for online algorithms to produce the optimal solution. The quality of the solution of an online algorithm is measured using a competitive ratio. It has been proven that for this problem, no online deterministic algorithm can achieve a competitive ratio better than $(2n - 1)$ and no online randomized algorithm can achieve an expected competitive ratio better than $\ln n$. It has been shown that advice in online computation improves the lower bound of the competitive ratio of online problems. Advice in online computation can be interpreted as additional information for the online algorithm to compensate for the lack of information about the whole input sequence. In this paper, we investigate how introducing machine-learned advice could improve the competitive ratio for this problem. We provide an online algorithm for the online assignment problem by simulating a machine learning (ML) algorithm that predicts the whole input in advance. We utilize an optimal offline algorithm to provide a matching solution from the predicted input. Furthermore, we investigate how the prediction error of ML affects the competitive ratio of the online algorithm. We utilize a benchmark data set to perform our empirical analysis of the solution quality. We show that as the ML prediction error increases, the solution quality decreases. Moreover, the magnitude of error is directly proportional to the size of the input. This result is analogous to the competitive ratio of the best deterministic algorithm for the online assignment problem which is dependent also on the parameter n. We show that our proposed online algorithm for the online assignment problem with ML prediction significantly outperforms the optimal deterministic online algorithm for the assignment problem. Also, we show that for some tolerable errors, i.e., for ML prediction with $RMSD(A, A') > 10$, our proposed online algorithm has a better solution quality. Otherwise, we propose the use of the existing best randomized online algorithm for the assignment problem. The trade-off for the solution quality is the

J. Caro et al. (eds.), *Proceedings of the Workshop on Computation: Theory and Practice (WCTP 2023)*, Atlantis Highlights in Computer Sciences 20,
https://doi.org/10.2991/978-94-6463-388-7_4

running time of the online algorithm which is highly dependent on the optimal offline algorithm and the ML predictor.

Keywords: online assignment problem, machine-learned advice, competitive analysis

1 Introduction

The *assignment problem* (AP) is a computational optimization problem that has diverse applications in various fields. Some of the few examples include assigning tasks to workers in a crowd-sourcing platform [1], passengers to vehicles in a taxi dispatching system [17,23], and receivers to access points in a wireless connection networks [9]. The problem is also known as the minimum-weighted bipartite matching problem. Given a complete bipartite graph, the goal is to obtain a matching with the minimum total edge weight. The most widely used algorithm to solve the problem is the *Hungarian algorithm* which runs at $O(n^3)$ time. An improvement of the algorithm for the assignment problem is from Karp et al. [12] which runs at $O(n^2 \log n)$ time.

Most combinatorial model of the problem assumes that the entire input is known before the start of the computation. However, most real-world problems deal with uncertainty. For instance, the entire input may not be known to the algorithm and given a partial input, the algorithm must decide or output a part of the solution. Problems with this limitation are called online problems. On the contrary, if the entire input is known, the problem is called an offline problem.

We focused on the online assignment problem variant that was introduced by Khuller et al [14]. This variant imitates real-world situations as data arrives with respect to time. Online algorithms for this variant must decide on what to do with the arriving nodes immediately and the decisions made are irrevocable. The drawback of online algorithms, however, is that they perform worse than their offline counterparts because of the lack of knowledge of the entire input sequence which leads to a less optimal solution. The *competitive ratio* of an online algorithm is used as a metric to compare the online algorithm to the optimal offline algorithm for the problem. For the assignment problem, the best deterministic algorithm is from [14] and [11]. It has a competitive ratio of $(2n-1)$ and is proven to be the tight lower bound for all online deterministic algorithms for the problem. In terms of expectation, an $O(\log^2 n)$-competitive randomized algorithm exists and has a tight lower bound of $O(\ln n)$ [2].

This paper is a continuation of our work in [5], where we provide an empirical analysis of the competitive ratio of the online assignment problem with respect to two parameters ϵ and μ that quantifies the difference between the actual and the predicted output. In the previous paper, ϵ quantifies the percentage of the total number of edges that differs in weight between the actual and predicted input. Meanwhile, the parameter μ quantifies the difference between the edge weights. In this study, we combined the two error parameters into a single one. We used the RMSD to represent the error between the original online input and the

predicted online input of the algorithm. Like what we have shown in the previous result in [5], we show that the RMSD value is directly proportional to the error in the solution quality. Thus, inversely proportional to the competitive ratio. We compare the empirical competitive ratio with the best deterministic online algorithm for the online assignment problem as well as the expected competitive ratio of the randomized online algorithms for the problem.

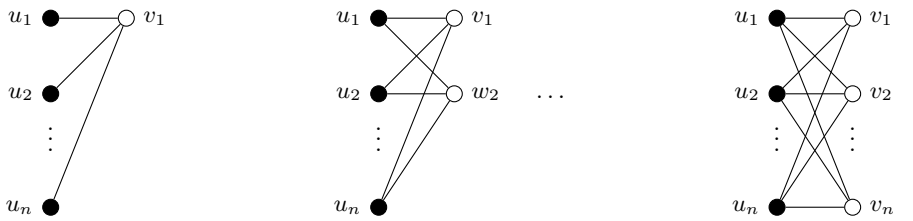
We will use the terms *AP* and *assignment problem*; *ML*, *machine learning*, and *machine learned*; *request*, *input* and *input sequence*; *competitive ratio* and *solution quality* will be used interchangeably.

2 Preliminaries

The formal definition of the *online assignment problem* is as follows.

Definition 1 (Online Assignment Problem) *Let $G = (U, V, E)$ be a complete bipartite graph, where U is a set of n black vertices and V is a set of n white vertices. Let the weight of an edge $(v_i, u_j) \in E$ be $e_{i,j} = d(v_i, u_j)$, where d is metric. At the onset, n black vertices are known, and as white vertex v_i arrives along with its edge weights, the algorithm matches u_i to one of the available black vertices. Once matched, a pair cannot later be separated. The algorithm tries to produce the smallest weight perfect matching possible.*

The complete bipartite graph is revealed in an online manner. Without loss of generality, we assume that the white vertices arrive from v_1 to v_n. We illustrate the online behavior in the following figure.



(a) t_1 (b) t_2 (c) t_n

Given this, there is a need to measure the goodness of an algorithm for the problem. In this paper, the *competitive ratio* will be used to measure the solution quality of the online algorithm, with respect to the solution obtained by the optimal offline algorithm. We illustrate the online behavior in the following figure.

Definition 2 (Competitive Ratio) *For all finite request sequences I, let* $\mathrm{ALG}(I)$ *be the worst-case performance of an online algorithm* ALG *and* $\mathrm{OPT}(I)$ *similarly to be the performance of an offline algorithm* OPT. ALG *has a competitive ratio of c (or is c-competitive) if there exists a constant b such that*

$$\mathrm{ALG}(I) \leq c \cdot \mathrm{OPT}(I) + b$$

If $b = 0$, ALG *has a strictly competitive ratio of c (or is strictly c-competitive) such that*

$$\mathrm{ALG}(I) \leq c \cdot \mathrm{OPT}(I)$$

2.1 Related Work

The pursuit of a faster and more efficient algorithm has always been an interest in the computer science research space. This is not different from what will be investigated in this paper, that is, to formulate an algorithm that results in a better solution quality for the online assignment problem. Some of the processes from Lykouris and Vassilvitskii's model will be followed, in which an online algorithm is merged with machine-learned advice [18]. These will be introduced later on.

Optimal Offline Algorithm. The constructed online algorithm, ALG, will involve using an optimal offline algorithm, OPT, for solving the online AP. Table 1 shows various algorithms that efficiently solve the offline AP.

Algorithm	Time Complexity
Kuhn, 1955 [15]	$O(n^4)$
Munkres, 1957 [20]	$O(n^3)$
Tomizawa, 1971 [25]	$O(n^3)$
Edmonds and Karp, 1972 [8]	$O(n^3)$
Karp, 1980 [12]	$O(n^2 \log n)$

Table 1: List of several offline algorithms for the AP and their time complexity

The *Hungarian Algorithm* is one of the best-known offline algorithms to solve the classical problem. [15] presented this algorithm, which was later refined by [20]. It was the first algorithm to solve the AP in a polynomial time, specifically, at $O(n^3)$ time. Other studies, such as [6], [25], and, [8], have also presented algorithms with the same time complexity.

In 1980, an $O(mn \log n)$ algorithm to solve the AP for m sources and n destinations was discovered by [12]. Its time complexity was achieved under the assumption that the costs of the edges are independent random variables and that the costs of the edges connected to a source are drawn independently from a common distribution.

Since this paper assumes that the sizes of the two disjoint sets are equal, [12] would have a running time of $O(n^2 \log n)$.

Online Algorithms. In this paper, known online algorithms for the AP will be used as a benchmark when comparing the solution quality of ALG. Table 2 shows various algorithms that efficiently solve the online AP.

The first known online version of an edge-weighted bipartite matching was introduced independently by [11] and [14]. In this version, assuming that the bipartite graph is complete, a set of vertices called *girl vertices* are given in advance, while the other set called *boy vertices* arrive one at a time. When a boy vertex arrives, he reveals the weights of edges connected to him and the girl vertices, and, he has to be matched off immediately, this decision is irrevocable. Similar to the Linear Sum Assignment Problem, the goal of this algorithm is to minimize the sum of the obtained weights.

Both papers, [11] and [14], gave a $(2n-1)$-competitive online algorithm to solve the problem and proved that no online deterministic algorithm can achieve a competitive ratio lower than that for all metric spaces.

Since the bound is proven tight, different approaches to achieve better solutions for the problem were done. One of these is using randomization as mentioned as an open problem in [11].

Furthermore, [19] discovered an online randomized algorithm with an expected competitive ratio of $O(\log^3 n)$ —the first of this kind to achieve a poly-logarithmic expected competitive ratio for the problem on general metrics. A year later, $O(\log^2 n)$-competitive randomized algorithm was discovered by [2], which improves the conversion from tree metrics to general metrics.

Table 2: List of several online algorithms for the AP and their competitive ratio

Algorithm	Competitive Ratio
Khuller et al., 1994 [14]	$2^n - 1$
Khuller et al., 1994 [14]	$2n - 1$
Meyerson et al., 2006 [19]	$\log^3 n$
Bansal et al, 2007 [2]	$\log^2 n$

In addition to comparing ALG's solution quality, its time complexity will also be determined.

Machine Learned Advice. An online algorithm that uses advice could make better responses to the requests arriving sequentially as it contains sufficient knowledge of the entire input sequence. This has been mentioned in [7] (oracle with answerer and helper modes), [4] (advice tape), and [24] (clairvoyant oracle with unlimited computational power).

A practical way to implement advice in real-life situations is through machine learning. [18] conceptualized a framework on how to utilize ML advice

to improve the performance of an online algorithm. Augmenting ML advice to online algorithms has been applied to multiple problems. [10] used ML advice to optimize the online page migration problem and discovered that the competitive ratio of their algorithm approaches 1 as the error rate of the predicted input diminishes to 0.

Lykouris et al. [18] and Rohatgi et al. [22] both tackled the caching problem and [16] and [21] with the ski-rental problem —all of which resulted in an improvement to the competitive ratios of their respective problems.

3 Online Algorithm for the Assignment Problem with ML Advice

Algorithm 1 shows the constructed online algorithm for the assignment problem, ALG, which is augmented with machine-learned advice. The questions considered in conceptualizing ALG were (1) whether an ML model could predict an input sequence close to the actual one, and, (2) whether an online algorithm could provide a solution or output close to the actual one by using the predicted input as a source of guidance.

Algorithm 1: Online Algorithm with ML Advice for the AP

input : Actual input A, an nxn matrix, where $a_{ij} \in \mathbb{N}^{[1,n]}$
output: Matching M

1 **for** $i \leftarrow 1$ ***to*** n **do**
2 | $a'_i \leftarrow mlModel(a_{i-1}, A)$; ▷ `returns predictions subject to error`
3 **end**
4 $P = Karp(A')$, where $P = p_1, p_2, ...p_n, p_i \in \mathbb{N}^{[1,n]}$ and is unique
5 **for** $i \leftarrow 1$ ***to*** n **do**
6 | $e_i \leftarrow (a_i, p_i)$
7 | $M \leftarrow M \cup \{e_i\}$
8 **end**

Lines 1-3 of Alg act as the reading section of the algorithm. It assumes that a certain ML model can be read line by line, for n times, which would result in an $n \times n$ when appended. The predicted matrix A' is a prediction matrix obtained from A differentiated by a certain defined degree of error. The method of how it is perturbed can be seen in 3.2.

The prediction matrix A' will be then used as an input to an optimal offline algorithm, [13] in this case, to get a matching P as the solution to A'

As it has been established that A' is a prediction for A, A' can be used as the actual input, and its matching P can be projected into A to obtain the solution M.

In a certain sense, this separates the Machine Learning Model from the algorithm. It makes sense that for this study, the ML Model is treated as a black box

that produces a prediction matrix A' subject to some error E. The quality of the ML model can be easily simulated using different values of E to describe how well the model predicts A' from A. With this, the study will revolve around empirical tests and analysis of the error metric E and the parameters that describe it to identify how such errors affect the solution quality of the online algorithm. The figure below shows the relationship and the parameters that will be used in this paper to investigate the effectiveness of this algorithm with the Online Assignment Problem.

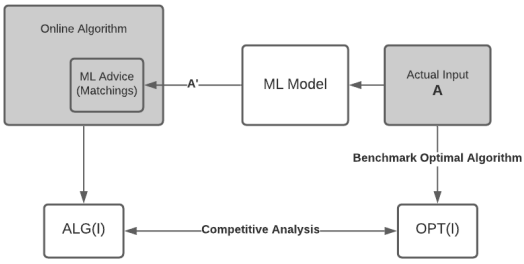


Fig. 1: Theoretical Framework for Algorithm 1 which shows the relationship between A and A' and how this paper compares the solution from a benchmark algorithm to Algorithm 1

This paper will use a Python library, *NetworkX*, to be able to use [12].

3.1 Running Time of the Algorithm

Line 4, in Algorithm 1, is [12] and it runs at $O(n^2 \log n)$ time, while lines 5-8 runs at $O(n)$ time as assigning an edge happens one at a time due to problem's manner of receiving input.

As mentioned earlier, ALG treats the ML model as a black box. Therefore, when the algorithm has been integrated with a certain machine learning model, the running time in lines 1-3 depends on what the model is and should be considered in determining the overall running time of ALG. Usually, more training time is needed to get better predictions.

Figure 2 shows a grouped bar chart of the running time obtained from functions, *graph creation* and *bipartite matching*, as the input size increases. Their execution time was measured using the benchmark data set from [3], and a computer using Macbook 2017 with a 2.5GHz dual-core Intel Core i7.

3.2 Obtaining predicted matrix A'

Since obtaining predictions from an ML model is beyond the scope of this study, the prediction or the ML advice will be based on matrix A, which will be the input to Algorithm 1.

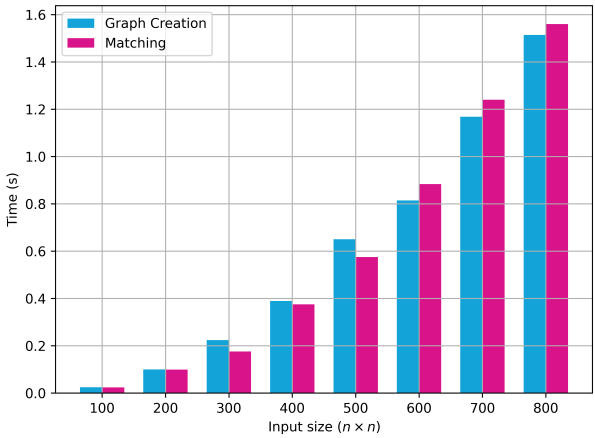


Fig. 2: Grouped bar chart of the execution time for (1) the creation of the bipartite graph, which was later used by [12], and (2) [12] itself. Both functions utilize a Python library, NetworkX. The actual running time was computed by executing the algorithm thrice using the dataset provided by [3]. Theoretically, (1) has a running time of $O(n^2)$ while (2) has a running time of $O(n^2 \log n)$. Therefore (2) is asymptotically faster than (1), which is evident in the figure when input size ≥ 600.

In this paper, it is assumed that there exists an *oracle* that somehow knows the actual input to the algorithm. This oracle can be visualized as a machine-learning model that predicts an input sequence. Specifically, this is the predicted matrix A', which is a perturbed version of A. Algorithm 2 shows how it is produced.

Aside from A, Algorithm 2 requires two parameters to proceed to the perturbation, ϵ and k. ϵ is a scale factor to determine the number of elements to perturb, while k is a constant to be used when perturbing a selected element.

To preserve the range of values of A to A', lines 4-10 solve this problem. Algorithm 2 has three conditions when perturbing an element:

1. If the value of the selected element exceeds the range of values in A if k is added, subtract k.
2. If the value of the selected element falls behind the range of values in A if k is subtracted, add k.
3. Otherwise, randomly choose whether the selected element should add or subtract k.

Equation 1 shows the closed-form solution for Algorithm 2.

$$A' = A + \text{RAND}(A|\epsilon) \times k_{ij} \tag{1}$$

In this equation, $\text{RAND}(A|\epsilon)$ return an $n \times n$ matrix in which, based on the scale factor ϵ, the selected elements will have a value of 1, and 0, otherwise.

Algorithm 2: Perturbation of Matrix A as an ML advice

input : Actual input A: an nxn matrix, where $a_{ij} \in \mathbb{N}^{[1,n]}$
ϵ: a scale factor to determine the number of elements to perturb
k: a value to be added or subtracted to the selected elements
output: Perturbed matrix A': an nxn matrix, where $a_{ij} \in \mathbb{N}^{[1,n]}$

Let min = minimum value in A, and max = maximum value in A
$A' = A$
Choose $\lfloor \text{size}(A) \times \epsilon \rfloor$ elements in A and do the following:
if $A_ij - k < min$ **then**
  $A'_ij = A'_ij + k$; `/* Case 1 */`
else if $A_ij + k > max$ **then**
  $A'_ij = A'_ij - k$; `/* Case 2 */`
else
  Randomly choose between Case 1 and Case 2.
end

Then, each element of this matrix will be scaled by k_{ij}, which depends on what the value of a_{ij} is. The range of values of k_{ij} is $\{-k, k\}$. Finally, adding the scaled matrix to A would result in a perturbed matrix A'.

The next section of the paper will discuss how the parameters ϵ and k, which are used to produce matrix A' as an ML advice, affect the solution quality of ALG, in other words, the correlation between the distance of A to A' and the competitive ratio of the algorithm.

Also, the succeeding section will discuss how far should A' be from A for the solution quality of Algorithm 1 to be better than the known bounds.

4 Results

An empirical test was conducted to determine how well Algorithm 1 performs in comparison to the known bounds for the algorithms of the assignment problem.

Instead of creating a dataset to be fed to ALG for the test, the study relied on a test dataset used in J.E. Beasley's "Linear programming on Cray supercomputers", as choosing a probabilistic distribution would not be a concern anymore.

The dataset contains 12 files, eight of which were appropriate for the testing. The selected files were then converted to $n \times n$ matrices in the implementation and were used as input and ML advice to Algorithms 1 and 2. The number of rows (or columns), n, of the matrices are $\{100, 200, 300, \ldots, 800\}$, and the range of values of the elements is from 1 to 100. The optimal solution of the matrices is also given in [3] and was verified to be correct during an examination.

Aside from the matrices of different sizes, Table 3 shows the sets of values for ϵ and k, which were used to produce an ML advice or a predicted matrix A'. The values for ϵ and k have been intentionally limited as the study assumes that when a machine learning model has been included in the algorithm in real

life, it is expected to produce accurate predictions before its training. Its trends and the analysis of these trends will be later identified in this paper.

Table 3: Selected values for ϵ and k. The study purposely selected 0.5 and 50 to be the largest values for the parameters ϵ and k, respectively, because it is assumed that an ML model can produce a matrix A' in which the number of elements to perturb will not reach 50% of the size of the matrix, and a selected element will not incur more than 50 from its original value.

Parameters	Values
ϵ	$\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$
k	$\{10, 30, 50\}$

Given all of these, Table 4 shows the resulting competitive ratios from every possible permutation of ϵ and k, using the selected $n \times n$ matrices, which are identified by the number of their rows, n.

It can be observed that when $\epsilon = 0$, for all k and n, the obtained competitive ratios were equal to 1. This observation is trivial because $\epsilon = 0$ means that the perturbation function has no elements to perturb from A. Thus, the predicted matrix A' is equal to the actual input A, and by the nature of ALG, its matching or solution will be the same as the solution obtained by A using any optimal offline algorithm for the AP.

Disregarding the values when $\epsilon = 0$, for each n, the lowest competitive ratios were from $\epsilon = 0$ and $k = 10$, while the highest was from $\epsilon = 0.5$ and $k = 50$. These values mean that the lower the number of chosen elements and the value for perturbation, for which ϵ and k are responsible, respectively, the lower the competitive ratio ALG will obtain. In general, a lower value for the parameters ϵ and k results in a lower prediction error, and thus, getting a better solution quality.

This prediction error could be in the form of a difference between the actual value and the prediction of the model. To get this difference, the *root-mean-square deviation* (RMSD) was used, it is one of the most commonly used to evaluate the performance of a regression model. Section 4.1 will present the definition of RMSD for this study and will discuss the relationship between the RMSD of A and A' and the solution quality of Algorithm 1.

4.1 Root-Mean-Square Deviation for A and A'

In this paper, Equation 2 shows the definition of the root-mean-square deviation. Its original formula has been adjusted for this study to accommodate $n \times n$

Table 4: Resulting competitive ratios of Algorithm 1 using the sets ϵ, k, and n. To get these values, the seed was set to 0 in the implementation.

ϵ	k	n							
		100	**200**	**300**	**400**	**500**	**600**	**700**	**800**
0	10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	30	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	50	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.1	10	1.2	1.6	2.0	2.3	2.7	2.8	3.0	3.2
	30	2.2	3.3	4.2	5.8	6.4	6.9	7.4	7.7
	50	4.7	7.6	7.9	11.4	12.7	14.6	15.4	16.5
0.2	10	1.6	2.1	2.7	3.2	3.5	3.6	3.7	4.1
	30	2.8	5.8	6.5	8.0	9.3	10.3	10.3	10.8
	50	6.1	11.6	13.3	16.3	18.2	19.6	20.6	21.9
0.3	10	1.9	2.7	3.4	3.6	4.1	4.1	4.4	4.5
	30	3.9	6.6	8.0	9.8	11.3	11.8	12.2	13.1
	50	7.7	14.0	17.4	19.8	21.3	21.9	23.0	23.9
0.4	10	2.1	2.9	3.7	4.2	4.4	4.5	4.7	4.8
	30	4.4	8.4	9.7	11.8	12.3	13.4	13.6	14.3
	50	9.6	15.4	18.8	22.2	23.0	24.1	24.9	25.1
0.5	10	2.6	3.2	4.2	4.4	4.7	4.8	5.0	5.1
	30	5.6	8.5	10.5	12.9	13.3	13.8	14.4	14.8
	50	11.6	17.3	21.0	23.3	23.9	25.0	25.1	25.5

matrices, in which it describes the distance of the value of each element of A' from A.

$$\mathrm{RMSD}(A, A') = \sqrt{\frac{\sum_{j=1}^{n}\sum_{i=1}^{n}(a_{ij} - a'_{ij})^2}{n^2}} \tag{2}$$

Using the equation, Table 5 shows the RMSD obtained from all permutations of ϵ and k for the 100×100 matrix and are appended with its corresponding competitive ratios.

It can be observed that the obtained RMSD values follow along the direction of the competitive ratios. For a certain ϵ, as the competitive ratio increases, the RMSD also increases. Furthermore, similar to the previous observation, the $\mathrm{RMSD}(A, A') = 0$ for all possible k values since $A = A'$.

To see what type of relationship they have, graphically, Figure 3 shows the scatter plot for the RMSD between A and A' versus the competitive ratios, which are listed in Table 5, along with a trend line. Ignoring the parameters, the values from the third and fourth columns of Table 5 are treated as y and x values, respectively, and each row is treated as an ordered pair, which were used as the data for the plot.

A linear trend line was included in the figure because it has the lowest R^2 value among any curved lines and polynomial lines of degree greater than 1. It has a positive slope and the scatter about the line appears to be small. Therefore,

Table 5: Resulting RMSD and competitive ratios using presented algorithm using the sets ϵ and k, and for the 100×100 matrix of the Beasley dataset.

ϵ	k	**RMSD**(A, A')	**Competitive Ratio**
0	10	0.0	1.0
	30	0.0	1.0
	50	0.0	1.0
0.1	10	3.2	1.4
	30	9.5	2.2
	50	15.8	4.7
0.2	10	4.5	1.8
	30	13.4	2.8
	50	22.4	6.1
0.3	10	5.5	2.0
	30	16.4	3.9
	50	27.4	7.7
0.4	10	6.3	2.1
	30	19.0	4.4
	50	31.6	9.6
0.5	10	7.1	2.3
	30	21.2	5.6
	50	35.4	11.6

RMSD(A, A') and the solution quality of the algorithm has a linear relationship with a strong and positive correlation. Moreover, a linear relationship means that both variables have a direct proportionality to each other.

5 Analysis

Looking at the trend of the competitive ratio with changing n and ϵ, it is quite easy to see its consistency with $RMSD(A, A')$ in which all variables k, ϵ, and n is directly proportional with the competitive ratio. We see in our results that for the smallest Beasley Data Set graph of size 100, the algorithm gives a competitive ratio of 1.344 on the lowest error benchmark for our empirical tests. The resulting competitive ratio then increases as we increase the error which further solidifies the hypothesis that with a better predictor, an algorithm for the online assignment problem with ML advice will result in an improved competitive ratio.

Comparing these results to the best deterministic and randomized algorithms, the online algorithm with advice performs considerably well when the prediction error is small. The graph below shows that the algorithm performs better than the best-randomized algorithms when $k <= 10$ and $\epsilon <= 0.5$ for all sizes of n in the Beasley Data set.

Now the analysis begs the question, *up to what extent does the online algorithm with advice perform better than its counterparts?* To answer this, a more

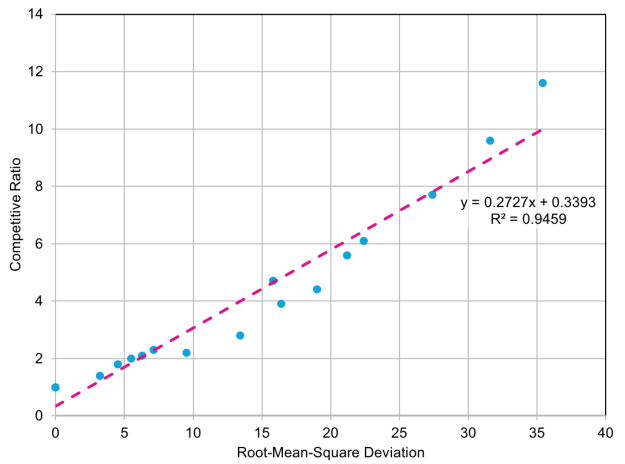


Fig. 3: Scatter plot of RMSD of the matrix of size 100×100 of the Beasley dataset, A, and the predicted matrix A', versus the obtained competitive ratio from ALG. The plot is also accompanied by a linear trend line, which best approximates the relationship between the two variables.

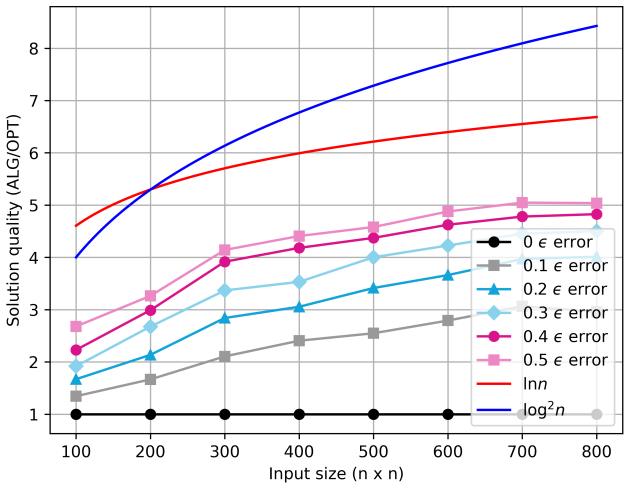


Fig. 4: Graphical comparison between competitive ratios of the presented algorithm and best known randomized algorithms with $k = 10$

representative measure should be used in place of the variables used in the previous graph. Figure 5 below uses the RMSD(A, A') as a measure of total error to show the performance of the online algorithm with advice.

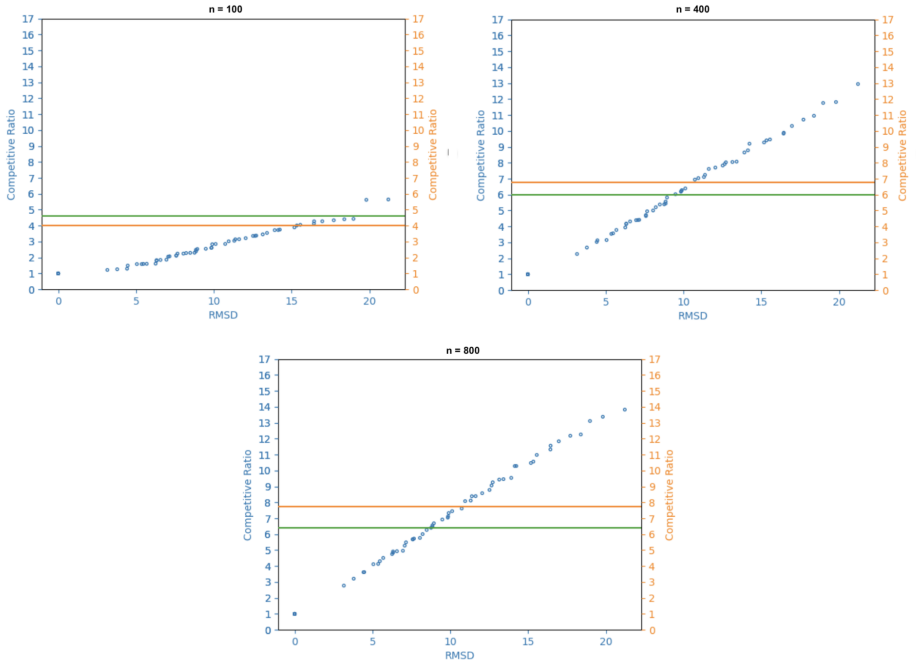


Fig. 5: Empirical competitive ratio obtained when increasing $RMSD(A, A)$. This plot provides a guide in identifying a tolerable RMSD threshold in deciding whether to incorporate ML in the online algorithm. For each plot, we can compare the performance of the best online randomized algorithm (orange) and known lower bound result (green) for $n = 100, 400, 800$, respectively.

It is shown in Figure 5 that certain $RMSD(A, A')$ values result in a better performance from the online algorithm with advice against the randomized algorithm. For a 100x100 graph, the algorithm will perform better as long as the $RMSD(A, A')$ does not exceed 3 and so on. Thus, as long as the prediction matrix A' is good enough to produce a small enough $RMSD(A, A')$, the online algorithm with ML Advice can perform better than the best deterministic and randomized algorithms.

Moreover, Figure 5 shows a range of tolerable RMSD for any ML predictor that will improve the current best competitive ratio. For RMSD greater than 10, we suggest utilizing randomized algorithms for the online assignment problem.

Even though the results have so far been positive with better predictions, one thing to consider for the algorithm is its trade-off with regard to the running time. As mentioned in the running time analysis section of this study, the running time of the ML Model pre-processing where the ML predictions are produced is highly dependent on the ML Model. This means that the performance of the algorithm improved with a worse running time as it is widely accepted that better

predictions from an ML model result from more training time (and data). This means that not in all cases that it would be advantageous that this algorithm will be used over the classical randomized and deterministic algorithms.

6 Future Work

Using ML advice for online algorithms is still an extremely young and underdeveloped approach to algorithm design. Even though this research has shown improvements against its classical counterparts, it is still a very fundamental approach to the problem and further research can still greatly improve its performance. With this, the researchers have identified several ideas to build upon the current research for the online assignment problem with ML Advice.

- A theoretical measure of the closed form of the function obtained from the algorithm that describes the relationship between the error and solution quality.
- Investigating the consistency of the presented algorithm when given a different benchmark data set.
- Investigating the consistency of the presented algorithm when given a data set with values from different metric spaces.
- Further analysis using different definitions of the error metric and/or different definitions of the perturbed matrix A'
- A less naive than the projection implementation of the algorithm.
- Real-world implementation of the algorithm using a Machine Learning Model to obtain the prediction matrix A'
- An empirical test in which a Machine Learning model is used to identify the specific trade-offs between an ML Model's running time and the competitive ratio of the solution.

References

1. Abdullah Alfarrarjeh, Tobias Emrich, and Cyrus Shahabi. Scalable Spatial Crowdsourcing: A Study of Distributed Algorithms. *Proceedings - IEEE International Conference on Mobile Data Management*, 1:134–144, sep 2015.
2. Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Seffi Naor. An o (log 2 k)-competitive algorithm for metric bipartite matching. In *European symposium on algorithms*, pages 522–533. Springer, 2007.
3. JE Beasley. Linear programming on cray supercomputers. *Journal of the Operational Research Society*, 41(2):133–139, 1990.
4. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *International Symposium on Algorithms and Computation*, pages 331–340. Springer, 2009.
5. Jhoirene B Clemente, Clarence Gabriel R Kasilag, and Pollux M Rey. Solving the Online Assignment Problem with Machine Learned Advice. In *Philippine Computing Science Congress*, 2021.
6. EA Dinic and MA Kronrod. An algorithm for the solution of the assignment problem. In *Soviet Math. Dokl*, volume 10, pages 1324–1326, 1969.

7. Stefan Dobrev, Rastislav Královič, and Dana Pardubská. How much information about the future is needed? In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 247–258. Springer, 2008.
8. Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.
9. Leong Hou U, Man Lung Yiu, Kyriakos Mouratidis, and Nikos Mamoulis. Capacity constrained assignment in spatial databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 15–27, 2008.
10. Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrović, and Ronitt Rubinfeld. Online page migration with ml advice. *arXiv preprint arXiv:2006.05028*, 2020.
11. Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.
12. Richard M Karp. An algorithm to solve the m× n assignment problem in expected time o (mn log n). *Networks*, 10(2):143–152, 1980.
13. Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.
14. 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.
15. Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
16. Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.
17. Der Horng Lee, Hao Wang, Ruey Long Cheu, and Siew Hoon Teo. Taxi Dispatch System Based on Current Demands and Real-Time Traffic Conditions:. *https://doi.org/10.3141/1882-23*, (1882):193–200, jan 2004.
18. Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *arXiv preprint arXiv:1802.05399*, 2018.
19. Adam Meyerson, Akash Nanavati, and Laura Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 954–959, 2006.
20. James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.
21. Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.
22. Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.
23. Kiam Tian Seow, Nam Hai Dang, and Der Horng Lee. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering*, 7(3):607–616, 2010.
24. Björn C Steffen. *Advice complexity of online graph problems*. ETH Zurich, 2014.
25. Nobuaki Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.