5-31-2023

A survey on online matching and ad allocation

Ryan Lee
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

A SURVEY ON ONLINE MATCHING AND AD ALLOCATION

by
Ryan Lee

One of the classical problems in graph theory is matching. Given an undirected graph, find a matching which is a set of edges without common vertices. In 1990s, Richard Karp, Umesh Vazirani, and Vijay Vazirani would be the first computer scientists to use matchings for online algorithms [8]. In our domain, an online algorithm operates in the online setting where a bipartite graph is given. On one side of the graph there is a set of advertisers and on the other side we have a set of impressions. During the online phase, multiple impressions will arrive and the objective of the online algorithm is to match incoming impressions to advertisers. The theory behind online matching is not only fascinating but has a lot of practical applications. One example is ridesharing platforms like Uber. An online algorithm can be used to assign incoming requests to available Uber drivers in order to maximize profits and fairness.

This paper will conduct a survey of online matching and ad allocation by focusing on four different problems: unweighted online matching, vertex weighted online matching, edge weighted online matching, and fairness maximization. Each problem will be discussed in three sections. In the first section, a detailed problem statement and the experimental setting will be given. The second section will go over different algorithms designed to tackle the problem. The third section will examine how the experiments were implemented and present the results. After each problem has been explored, a comparison of the four problems will be given.

A SURVEY ON ONLINE MATCHING AND AD ALLOCATION

by
Ryan Lee

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Data Science

Ying Wu College of Computing
Department of Data Science

May 2023

APPROVAL PAGE

A SURVEY ON ONLINE MATCHING AND AD ALLOCATION

Ryan Lee

Dr. Pan Xu, Thesis Advisor, Dissertation Advisor — Date
Assistant Professor of Computer Science, NJIT

Dr. Zuofeng Shang, Committee Member, Committee Member — Date
Associate Professor of Mathematical Sciences, NJIT

Dr. James Geller, Committee Member, Committee Member — Date
Professor of Data Science, NJIT

BIOGRAPHICAL SKETCH

Author: Ryan Lee

Degree: Master of Science

Date: May 2023

Undergraduate and Graduate Education:

- Bachelor of Science in Computer Science,
  Rutgers University - New Brunswick, New Brunswick, NJ, 2019
- Minor in Mathematics,
  Rutgers University - New Brunswick, New Brunswick, NJ, 2019
- Certificate in Computer Science,
  Arizona State University, Tempe, AZ, 2020
- Master of Science in Data Science,
  New Jersey Institute of Technology, Newark, NJ, 2023

Major: Data Science

I would like to dedicate this thesis to my parents Jiang Huining and Li Zhonglu. Thank you for always supporting me.

Ryan Lee

ACKNOWLEDGMENT

I want to express gratitude towards my advisor Dr. Pan Xu for being a great advisor and a great friend.

I also want to thank the Dr. James Geller and Dr. Zuofeng Shang for their guidance.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION 1

1.1 Overview of Online Matching and Ad Allocation 1

1.2 Graph Theory Basics 2

2 UNWEIGHTED ONLINE MATCHING 6

2.1 Problem Statement 6

2.2 Algorithms and Theoretical Results 8

2.3 Experimental Design and Results 12

3 EDGE WEIGHTED ONLINE MATCHING 16

3.1 Problem Statement 16

3.2 Algorithms and Theoretical Results 16

3.3 Experimental Design and Results 18

4 VERTEX WEIGHTED MATCHING 22

4.1 Problem Statement 22

4.2 Algorithms and Theoretical Results 22

4.3 Experimental Design and Results 27

5 FAIRNESS MAXIMIZATION 31

5.1 Problem Statement 31

5.2 Algorithms and Theoretical Results 32

5.3 Experimental Design and Results 33

6 COMPARISON 45

6.1 Comparing Problem Statements 45

6.2 Comparing Algorithms and Theoretical Results 46

BIBLIOGRAPHY 48

LIST OF TABLES

Table		Page
2.1	SM and TSM Algorithms on Varying δ with $\|A\| = 100, \|I\| = \|T\| = 500$.	13
2.2	SM and TSM Algorithms on Varying $\|T\|$ with $\|A\| = \|I\| = 100$ and $\delta = 3$	14
2.3	SM and TSM Algorithms on Varying $\|A\|$ with $\|I\| = \|T\| = 300$ and $\delta = 3$	14
3.1	HMZ 1 and HMZ 2 Algorithms on Varying δ with $\|I\| = 100, \|J\| = \|T\| = 500$	19
3.2	HMZ 1 and HMZ 2 Algorithms on Varying $\|T\|$ with $\|I\| = \|J\| = 100$ and $\delta = 3$	20
4.1	Ranking and Greedy-Perturbed Algorithms on Varying δ with $\|U\| = \|V\| = \|T\| = 100$	28
4.2	Ranking and Greedy-Perturbed Algorithms on Varying $\|T\|$ with $\|U\| = \|V\| = 100$ and $\delta = 3$	29
5.1	Varying δ on IFM with fixed T, $\|I\| = 100, \|J\| = \|T\| = 100$	35
5.2	Varying δ on IFM with fixed T, $\|I\| = 100, \|J\| = \|T\| = 500$	35
5.3	Varying T on IFM with fixed $\delta = 2$, $\|I\| = 100, \|J\| = \|T\|$	37
5.4	Varying T on IFM with fixed $\delta = 3$, $\|I\| = 100, \|J\| = \|T\|$	37
5.5	Varying δ on VOM CR1, $\|I\| = \|J\| = \|T\| = 100$	39
5.6	Varying δ on VOM CR2, $\|I\| = \|J\| = \|T\| = 100$	39
5.7	Varying T on VOM CR1 with $\delta = 3$, $\|I\| = \|J\| = \|T\|$	41
5.8	Varying T on VOM CR2 with $\delta = 3$, $\|I\| = \|J\| = \|T\|$	41

LIST OF FIGURES

Figure		Page
1.1	Two simple graphs.	2
1.2	An example of a cycle and a path.	3
1.3	An example of a bipartite graph and a maximum matching.	4
2.1	Example Graph.	7
2.2	SM and TSM Algorithms on Varying δ with $\|A\| = 100, \|I\| = \|T\| = 500$.	13
2.3	SM and TSM Algorithms on Varying $\|T\|$ with $\|A\| = \|I\| = 100, \delta = 3$.	14
2.4	Varying $\|A\|$ with $\|I\| = \|T\| = 300, \delta = 3$.	15
3.1	HMZ 1 and HMZ 2 Algorithms on Varying δ with $\|U\| = 100, \|V\| = \|T\| = 500$.	20
3.2	HMZ 1 and HMZ 2 Algorithms on Varying $\|T\|$ with $\|I\| = \|J\| = 100, \delta = 3$.	21
4.1	Ranking and Greedy-Perturbed on Varying δ with $\|U\| = \|V\| = \|T\| = 100$.	28
4.2	Ranking and Greedy-Perturbed on Varying $\|T\|$ with $\|U\| = \|V\| = 100$ and $\delta = 3$.	29
5.1	Varying δ on IFM with fixed T, $\|I\| = 100, \|J\| = \|T\| = 100$.	36
5.2	Varying δ on IFM with fixed T, $\|I\| = 100, \|J\| = \|T\| = 500$.	36
5.3	Varying T on IFM with fixed $\delta = 2$, $\|I\| = 100, \|J\| = \|T\|$.	38
5.4	Varying T on IFM with fixed $\delta = 3$, $\|I\| = 100, \|J\| = \|T\|$.	38
5.5	Varying δ on VOM CR1, $\|I\| = \|J\| = \|T\| = 100$.	40
5.6	Varying δ on VOM CR2, $\|I\| = \|J\| = \|T\| = 100$.	40
5.7	Varying T on VOM CR1 with $\delta = 3$, $\|I\| = \|J\| = \|T\|$.	42
5.8	Varying T on VOM CR2 with $\delta = 3$, $\|I\| = \|J\| = \|T\|$.	42

CHAPTER 1

INTRODUCTION

1.1 Overview of Online Matching and Ad Allocation

Online Matching is a fairly recent problem that has roots in classical fields like graph theory which stretch back decades ago. Consider a bipartite graph $G(A, I, E)$, where A is the set of advertisers, I is the set of impressions, and E is the set of edges. One can think of the impressions as agents who make requests and think of advertisers as agents who take requests. The impressions will arrive one by one in sequential fashion and must be immediately matched to an available advertiser in A by an algorithm. The algorithm's objective is to maximize the number of matchings and this is known as the online bipartite matching problem. In order to evaluate an algorithm's performance, the competitive ratio is used which is the ratio between the performance of the algorithm and the offline optimal.The first algorithm introduced for this problem was the ranking algorithm created by Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani in the 1990s [8]. The ranking algorithm achieved a competitive ratio of approximately $1 - \frac{1}{e}$. Since then a number of new algorithms have been created with a higher competitive ratio.

Algorithms designed for online bipartite matching have a wide variety of practical applications. The most notable application is ad allocation. Since the rise of the internet, advertisers will pay websites or ad networks to display their ads which is typically done through contracts or bids. This dynamic can be modeled as a matching problem where the goal is to optimize the matching. Unfortunately, traditional optimization techniques are not practical due to two reasons. The first reason is that the size of online matching problems are too big to be handled by a simple linear programming model. The second reason is that the order of the

(a) Undirected Graph. **(b)** Directed Graph.

Figure 1.1 Two simple graphs.

impressions are not known beforehand. Thus, new approaches had to be created in order to deal with this issue. Other applications for online matching include ride sharing platforms such as Uber or Lyft, where an algorithm has to match incoming requests with available drivers. Fairness maximization is also another application, where the goal is to make sure that the number of matchings is spread out between different groups.

1.2 Graph Theory Basics

A graph $G(V, E)$ is a discrete structure that contains points called vertices which are denoted by V. Edges are lines that connect vertices together and are denoted by E. A graph is called directed when the edges have a direction. The converse is called an undirected graph. This means that in a directed graph, an algorithm can only move in the direction that an edge is pointing towards. For undirected graph, an algorithm can move in any direction for any given edge. Figure 1.1, shows an example. On the left is a undirected graph which has no arrows on it's edges. On the right is a directed graph with arrows indicating direction.

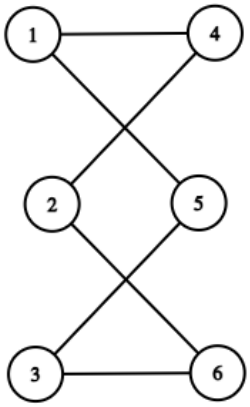
Two vertices are called adjacent if there is an edge connecting them. Two edges

(a) Graph Cycle. **(b)** Graph Path.

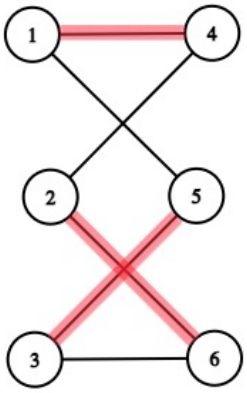
Figure 1.2 An example of a cycle and a path.

are called adjacent if they share a vertex. A path is a finite or infinite sequences of edges which connects a sequence of vertices. A cycle is path that starts and ends with the same node. The length of a path or cycle is the number of edges in it. A simple graph is a graph that does not have more than one edge between any two vertices and does not contain cycles. Figure 1.2, contains an example of a cycle and path. On the left is a cycle that goes from node one to node two to node three and back to node one. On the right, is a path that goes from node one to node two to node three and ends at node four.

A bipartite graph is an undirected graph that contains two sets of distinct vertices such that there does not exist an edge between members in the same set. Bipartite graphs contain no odd-numbered cycles, because if one did exist that would imply an edge between two vertices of the same set. This would contradict the definition of a bipartite graph. In undirected graphs, a matching is a set of edges that share no common vertices. In other words, every edge in a matching is non-adjacent and there are no cycles. A maximal matching is a matching that is not a subset of any other matching. A maximum matching is a matching that contains the highest number of edges. Every maximum matching is maximal, but the converse is not



(a) Bipartite Graph.



(b) Maximum Matching.

Figure 1.3 An example of a bipartite graph and a maximum matching.

true. Additionally, a graph can contain multiple maximal matchings and multiple maximum matchings. Figure 1.3 contains an example of a bipartite graph and a maximum matching. On the left, there is a bipartite graph with six nodes. Each node has two edges neighboring it, and the nodes are divided into two groups with no edges between members of the same group. On the right is the same graph but now there is a maximum matching which is colored red. This matching is maximum because if anymore edges are colored red then some of the vertices will share an edge. This would contradict the definition of a matching.

A flow network is a directed graph $G(V, E)$ that contains a source node s and a sink node t. Every edge has a capacity c which describes how much flow f can go through that edge. The capacity c is a nonnegative real number but most of the algorithms in this paper treat c as a nonnegative integer. The flow f can be represented as the function $f : V \times V \longrightarrow \mathbb{Z}^+$. If the flow for a particular edge is 0, it implies that the edge does not exist. Every flow network must satisfy three properties:

1. The flow going through an edge cannot be greater than the capacity of the edge: $\forall v_1, v_2 \in V,\ f(v_1, v_2) \leq c(v_1, v_2)$.

2. The total flow going into a node must equal the total flow leaving the node: $\forall v_2 \in V, \sum_{(v_1,v_2)\in E} f(v_1, v_2) = \sum_{(v_2,v_3)\in E} f(v_2, v_3)$.

3. The flow going from v_1 into v_2 is equal to the negative flow going from v_2 to v_1: $\forall v_1, v_2 \in V,\ f(v_1, v_2) = -f(v_2, v_1)$.

The maximum flow problem is to find the maximum possible flow from the source node s to the sink node t. Many different algorithms exist to solve this problem and the most well known one is the Ford-Fulkerson algorithm. But this paper will use the Preflow-Push algorithm instead, the details of which will be left out and instead referenced.

CHAPTER 2

UNWEIGHTED ONLINE MATCHING

2.1 Problem Statement

Let $G(A, I, E)$ be a bipartite graph where A is the set of advertisers, I is the set of impressions, and E is the set of edges. In figure 2.1, an example of this graph is given. The set of edges and vertices are unweighted and the impressions arrive in sequential order. When an impression $i \in I$ arrives, i has to be assigned to an advertiser $a \in A$ such that there exists an edge $e \in E$ between i and a. The set of advertisers A are fixed and known before an online algorithm is executed. Each advertiser to only be assigned at most once. When it comes to I and E it largely depends on the setting the online algorithm operates in. There are two settings that are commonly examined: *adversarial* and *iid*. In the adversarial setting the online algorithm will have no information about I and E beforehand. In the *iid* (identically and independently distributed) setting, the edges E and the probability distribution D will be known beforehand. The setting this section uses is *iid*.

The probability distribution D describes the likelihood an impression $i \in I$ will arrive. More formally, given a probability distribution D the probability that an impression i will arrive is $P(i) = \frac{e_i}{n}$, where e_i is the expected number of times an impression i arrives and n is the total number of impressions that will arrive [4]. Clearly $n = \sum_{i \in I} e_i$. The objective of the algorithm will then be to maximize the number of matchings on the realization graph $\hat{G}(\hat{I})$ which is a graph that is based on the probability distribution D. Formally, the realization graph $\hat{G}(\hat{I})$ is a bipartite graph $\hat{G}(A, \hat{I}, \hat{E})$ where $\hat{I} = \{\hat{i}_k : \hat{i}_k \in D(i), k = 1, 2, \ldots, e_i, \forall i \in I\}$ and $\hat{E} = \{(a, \hat{i}) : a \in A, \hat{i} \in \hat{I}\}$. For unweighted online matching there are two phases: the offline phase and the online phase. These two phases largely depend on the algorithm

being used but one can think of the offline phase as the "planning phase" and the online phase as the "execution phase."

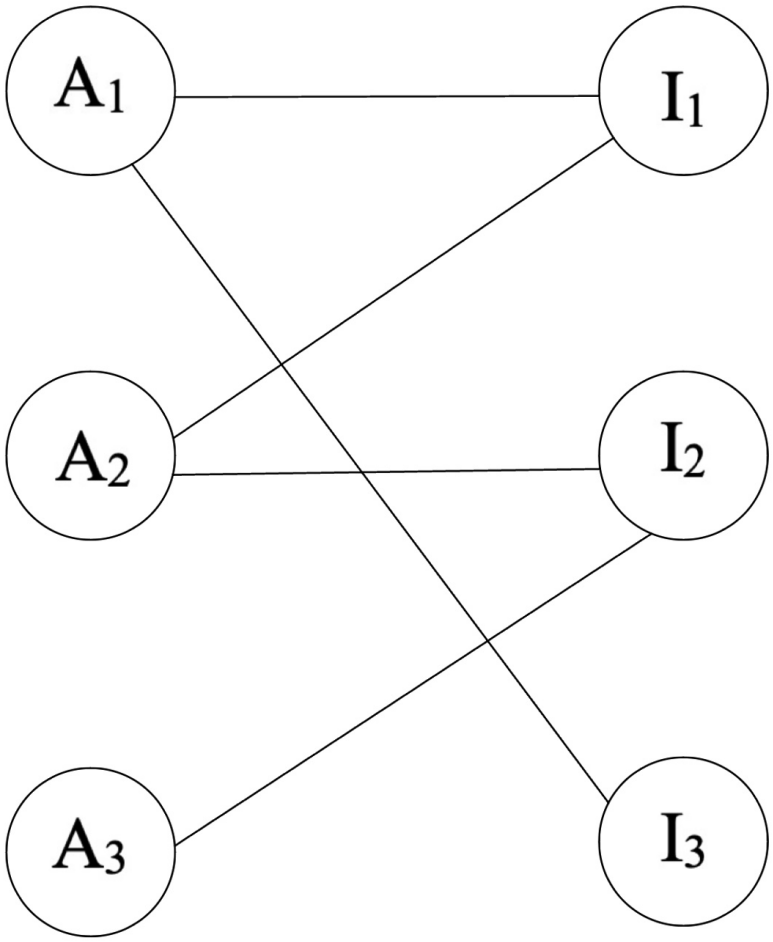


Figure 2.1 Example Graph.

Now the problem statement can be condensed. Let $\Gamma = (G, D, n)$ be a particular instance where $G = (A, I, E)$ is bipartite graph, D is a probability distribution, and n is the total number of impressions [4]. During the offline phase, construct the realization graph $\hat{G}(\hat{i})$ and create a "plan." During the online phase, the algorithm's objective is to attempt to assign each impression i to an advertiser a based on the "plan." The goal is the maximize the number of assignments, or in more mathematical terms the goal is to have $\frac{ALG(\hat{I})}{OPT(\hat{I})} \geq \alpha$ with high probability. $\frac{ALG(\hat{I})}{OPT(\hat{I})}$ is the competitive

ratio between the performance of the algorithm and offline optimal. α is the approximation ratio [4].

2.2 Algorithms and Theoretical Results

There are two important mathematical facts that needs to be established:

1. Suppose n balls are thrown into n bins, i.i.d with uniform probability over the bins. Let B be a particular subset of the bins, and S be a random variable that equals the number of bins from B with at least one ball. For any $\epsilon > 0$, the probability that $|B|(1-\frac{1}{e})-\epsilon n \leq S \leq |B|(1-\frac{1}{e}+\frac{1}{en})+\epsilon n$ is at least $1-2e^{-\epsilon n/2}$.

2. Suppose n balls are thrown into n bins, i.i.d with uniform probability over the bins. Let B_1, B_2,..., B_ℓ be ordered sequences of bins, each of size c, where no bin is in more than d such sequences. Fix some arbitrary subset $R \subseteq \{1,\ldots,c\}$. A bin sequence $B_a = (b_1, ..., b_c)$ is considered "satisfied" if (i) at least one of its bins b_i with $i \notin R$ has at least one ball in it; or, (ii) at least one of its bins b_i with $i \in R$ has at least two balls in it. Let S be a random variable that equals the number of satisfied bin sequences. The probability that $S \geq \ell(1-\frac{2^{|R|}}{e^c}) - \epsilon dn - \frac{2^{|R|}c^2}{e^c}\frac{\ell}{n-c^2}$, is at least $1-2e^{-\epsilon^2 n/2}$.

The proofs of these two facts are not included and were proved in and established in [4]. The first algorithm that this section examines is the "Suggested Matching" Algorithm or SM Algorithm for short. It was introduced by Jon Feldman, Aranyak Mehta, Vahab Mirrokni, S. Muthukrishnan. Below is the pseudocode for the SM Algorithm:

In order to analyze the theoretical performance of the SM algorithm, the competitive ratio $\frac{ALG(\hat{I})}{OPT(\hat{I})}$ must be bounded. The following is a reworded summary of an analysis conducted in [4]. Let $\Gamma = (G, D, n)$ be a particular instance of the problem. Let $F_a = \sum_i f_{ai}$, where f_{ai} is the flow going from advertiser a to impression

Algorithm 1: "Suggested Matching" Algorithm (SM Algorithm) [4].

Offline Phase:

Let $\Gamma = (G, D, n)$ be a particular instance of the problem.

Using Γ construct the flow network graph G_f as follows:

Create a new source node s and for every advertiser $a \in A$ create a directed edge with going from s to a with capacity 1.

Create a new sink node t and for every impression $i \in I$ create a directed edge going from i to t with capacity e_i.

Orient all edges in G to go from A to I and let the capacity of these edges be 1.

Find the max flow f on resulting graph.

Online Phase:

for $t = 1, \ldots, n$ **do**

  Let an impression i' arrive at time t according to the probability distribution D.

  Choose a random ad a' with probability $\frac{f_{a'i}}{e_i}$.

  If a' is available match a' with i' otherwise reject i'.

i. Since each advertiser a can only be matched at most once, then F_a is either 0 or 1. In other words, F_a indicates whether a was matched. Let $A^* = \{a \in A : F_a = 1\}$, which is the set of matched advertisers. For any $a \in A^*$, since at least $f_{ai} = 1$ then the probability the advertiser a gets chosen for that particular impression i is $\frac{1}{e_i}$. The probability i gets chosen is $\frac{e_i}{n}$. Therefore the probability that $a \in A^*$ gets chosen is $\frac{1}{e_i} \cdot \frac{e_i}{n} = \frac{1}{n}$. The probability that $a \in A^*$ gets chosen at least once is $(1-(1-\frac{1}{n})^n) \approx 1-\frac{1}{e}$. Clearly this can be modeled as a balls-in-bin problem so using fact 1, the probability that $ALG \geq (1-\frac{1}{e})|A^*| - \epsilon n$ is at least $1 - e^{-\Omega(n)}$.

Let G_f be the flow network graph and G_r be the residual graph after finding

the max flow. S is the set of nodes reachable from s in the residual graph. T follows a similar definition. Then $A_S = A \cap S$ and define A_T, I_S, I_T similarly. Let $e' \in E$ be an edge that goes from some $a' \in A_S$ to some $i' \in I_T$. Since $a' \in A_S$ and there is an edge e' that goes from a' to i', then $i' \in I_S$. But $i' \in I_T$, which implies flow went from i' to t. This implies flow went from a' to i', which then implies flow went from s to a'. If flow went from s to a' then a' should not be reachable in the residual graph. This is a contradiction. Thus edges from A_S to I_T cannot exist. The only edges that can exist are from s to A_T and from I_S to t. Therefore $|A^*| = \sum_a F_a = |A_T| + \sum_{i \in I_S} e_i$.

Let $\hat{G}$ be the realization graph where the capacity of every edge is 1. Let $\hat{G}_f$ be the network-flow graph on $\hat{G}$. Define the $s - t$ cut in $\hat{G}_f$ as follows: $\hat{I}_S = \cup_{i \in I_S} D(i)$, $\hat{I}_T = \cup_{i \in I_T} D(i)$, $\hat{S} = A_S \cup \hat{I}_S$, $\hat{T} = A_T \cup \hat{I}_T$. Using similar reasoning for G_r, there are no edges from A_S to $\hat{I}_T$ in $\hat{G}_r$. Therefore the size of this cut is $|\hat{I}_S| + |A_T|$ and the probability an impression i is in $\hat{I}_S$ is $\sum_{i \in I_S} \frac{e_i}{n}$. Since the OPT is bounded by the size of this cut, using a Chernoff bound the probability that $OPT \leq |A_T| + \sum_{i \in I_S} e_i + \epsilon n = |A^*| + \epsilon n$ is at least $1 - e^{-\Omega(n)}$.

Since $OPT \leq |A^*| + \epsilon n$ and $(1 - \frac{1}{e})|A^*| - \epsilon n \leq ALG$ then multiplying the two inequalities give $OPT \cdot ((1 - \frac{1}{e})|A^*| - \epsilon n) \leq (|A^*| + \epsilon n) \cdot ALG$. Rearranging the terms gives $\frac{(1-\frac{1}{e})|A^*| - \epsilon n}{|A^*| + \epsilon n} \leq \frac{ALG}{OPT}$. Let ϵ be a small number close to zero, then $\frac{(1-\frac{1}{e})|A^*| - \epsilon n}{|A^*| + \epsilon n} \approx \frac{(1-\frac{1}{e})|A^*|}{|A^*|} = (1 - \frac{1}{e})$. Therefore with high probability, $(1 - \frac{1}{e}) \leq \frac{ALG}{OPT}$.

The SM algorithm performs well with a approximation ratio of at least $(1 - \frac{1}{e})$, but the "Two Suggested Matchings" Algorithm, also known as the TSM algorithm, builds upon the SM algorithm and improves the approximation ratio. Created by the same authors who created the SM algorithm, the pseudo-code is given below.

It has been proved that the TSM algorithm achieves an approximation ratio of roughly $\frac{1-\frac{2}{e^2}}{\frac{4}{3}-\frac{2}{3e}} \approx 0.67029$ [4]. This improves on the SM algorithm with an approximately 0.04 boost in performance. This proof is similar to the analysis

Algorithm 2: "Two Suggested Matchings" Algorithm (TSM Algorithm) [4].

Offline Phase:

Let $\Gamma = (G(A, I, E), D, n)$ be a particular instance of the problem.

Using Γ construct the boosted flow network graph G_f as follows:

Create a new source node s and for every advertiser $a \in A$ create a directed edge with going from s to a with capacity 2.

Create a new sink node t and for every impression $i \in I$ create a directed edge going from i to t with capacity 2.

Orient all edges in G to go from A to I and let the capacity of these edges be 1.

Find the max flow f on resulting graph.

Let E_f be the set of edges $(a, i) \subseteq E$ with non-zero flow and treat these edges as undirected.

Now color the edges of E_f with the coloring algorithm given in [4].

Online Phase:

for $t = 1, \ldots, n$ **do**

    Let an impression i' arrive at time t according to the probability distribution D.

    If this is the first time i' is arriving, assign it to the advertiser on the blue edge if that advertiser is available.

    If this is the second time i' is arriving, assign it to the advertiser on the red edge if that advertiser is available.

    Reject otherwise.

conducted above but also makes use of fact 2. Due to the length of the proof, the proof has been left out and instead referenced.

2.3 Experimental Design and Results

Three experiments were done in this section. The first experiment fixes the number of advertisers to be 100, and fixes the arrival size and impression size to both be equal to 500. The number of neighbors, which is denoted by δ, will be varied from 1 to 6. In the second experiment, the number of advertisers and the number of impressions are both fixed to 100 and δ is fixed to 3. The arrival size will be varied using values from this set $\{50, 100, 300, 500, 800, 1000\}$. The third experiment fixes the number of impressions and the arrival size to be 300 and fixes δ to be 3. This time the number of advertisers will be varied using values from the set $\{50, 100, 150, 200, 250, 300\}$.

For each experiment, a synthetic graph is generated along with a synthetic arrival sequence. The synthetic graph is generated by randomly picking δ neighbors for each advertiser in A. For each neighbor, an edge is generated between that neighbor and the advertiser. An arrival sequence is generated by uniformly picking $|T|$ impressions from I with replacement. Afterwards, the probability distribution is computed by counting the number of times each impression appears in the arrival sequence. After all synthetic data is generated, the SM and TSM algorithms are run in each experimental setting and the performance is recorded. The OPT is calculated using the following linear program [5], where $a \sim i$ indicates i is a neighbor of a. Then the competitive ratio is calculated by dividing the algorithm performance by the OPT. The results are shown below.

$$\max \sum_{a \in A} \sum_{a \sim i} X_{ai} \text{ subject to:} \tag{2.1}$$

$$\forall a \in A : \sum_{a \sim i} X_{ai} \leq 1 \tag{2.2}$$

$$\forall i \in I : \sum_{i \sim a} X_{ai} \leq 1 \tag{2.3}$$

$$\forall e \in E : 0 \leq x_e \tag{2.4}$$

Table 2.1 SM and TSM Algorithms on Varying δ with $|A| = 100, |I| = |T| = 500$

DELTA	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$
SM	0.64	0.80	0.74	0.74	0.71	0.66
TSM	0.09	0.30	0.26	0.16	0.26	0.25

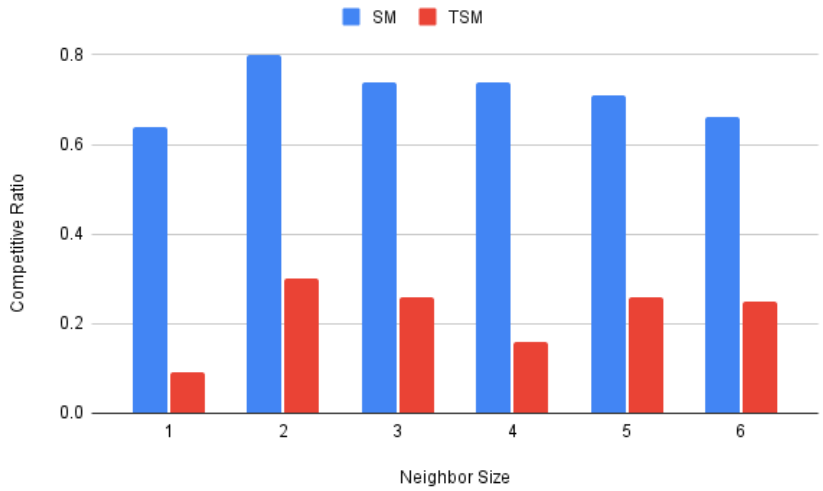


Figure 2.2 SM and TSM Algorithms on Varying δ with $|A| = 100, |I| = |T| = 500$.

The performance has a lot of variation. For the varying δ experiment, surprisingly the SM algorithm is outperforming the TSM algorithm considerably. For the TSM algorithm, increasing the number of neighbors from 1 to 2 increases the performance. But the performance stops increasing as δ increases past 2. In the varying $|T|$ experiment, both the SM and TSM algorithms under perform initially. But as $|T|$ increases, the performance of the TSM algorithm increases substantially while the performance of the SM algorithm increases initially and then starts to stagnate. For

Table 2.2 SM and TSM Algorithms on Varying $|T|$ with $|A| = |I| = 100$ and $\delta = 3$

ARRIVALS	$t = 50$	$t = 100$	$t = 300$	$t = 500$	$t = 800$	$t = 1000$
SM	0.30	0.45	0.44	0.44	0.46	0.46
TSM	0.39	0.64	0.92	0.93	0.95	0.97

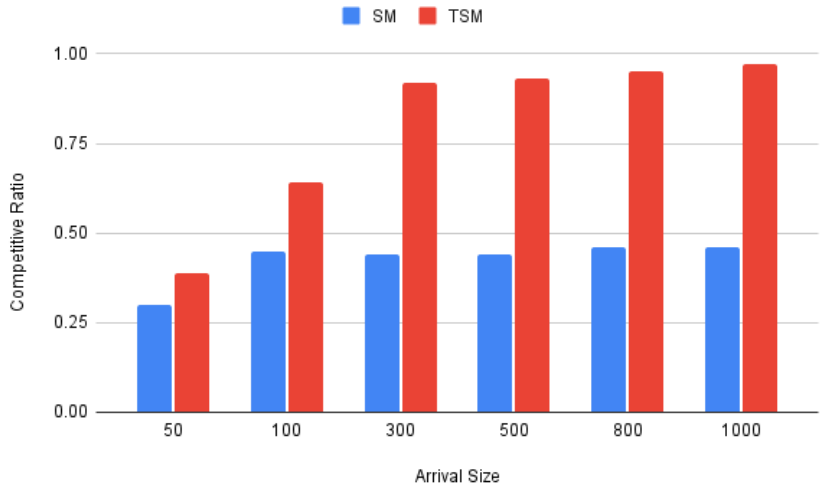


Figure 2.3 SM and TSM Algorithms on Varying $|T|$ with $|A| = |I| = 100, \delta = 3$.

Table 2.3 SM and TSM Algorithms on Varying $|A|$ with $|I| = |T| = 300$ and $\delta = 3$

ADVERTISERS	$i = 50$	$i = 100$	$i = 150$	$i = 200$	$i = 250$	$i = 300$
SM	0.74	0.64	0.67	0.52	0.50	0.42
TSM	0.39	0.45	0.57	0.63	0.64	0.65

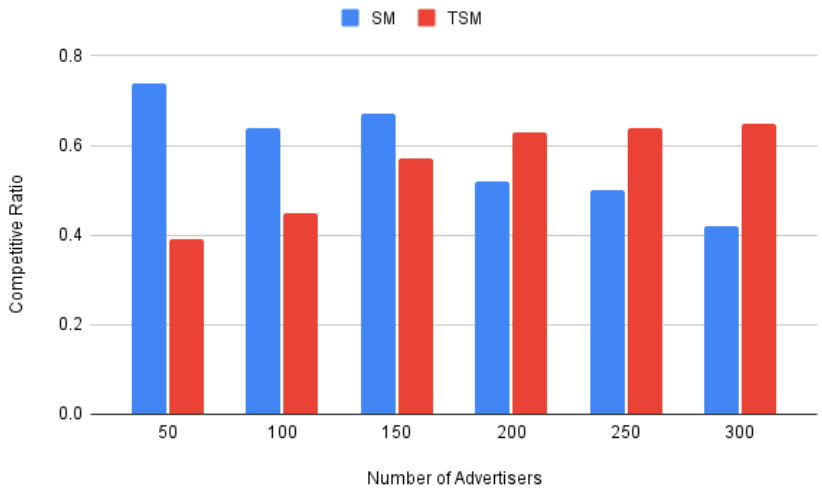


Figure 2.4 Varying $|A|$ with $|I| = |T| = 300, \delta = 3$.

the last experiment where $|A|$ varies, the SM algorithm initially outperforms the TSM algorithm. But as the number of the advertisers increase, the TSM algorithm performance goes up while the SM algorithm performance goes down. Eventually the TSM algorithm surpasses the SM algorithm performance. The reason for this performance is not entirely clear but one possible explanation is that the SM algorithm makes use of the probability distribution in the boosted flow graph, while the TSM algorithm does not. This means when the number of impressions overwhelms the number of advertisers, the TSM algorithm suffers because it does not make use of the distribution. On the other hand, the SM algorithm is guided by the distribution and can distribute resources better. This seems to be supported by the fact that as the number of advertisers approaches the number of impressions, the TSM algorithm performance increases and outperforms the SM algorithm in accordance with the theoretical results.

CHAPTER 3

EDGE WEIGHTED ONLINE MATCHING

3.1 Problem Statement

Let $G(A, I, E)$ be a bipartite graph, where A is the set of advertisers, I is the set of impressions, and E is the set of edges. Figure 2.1 contains an example of this graph for visualization purposes. Similar to the unweighted setting, impressions arrive in sequential order and must be assigned to an advertiser $a \in A$ such that there exists an edge $e \in E$ between i and a. The key difference is that every edge $e \in E$ has a weight associated with it denoted by $\omega(e)$. The edge-weighted setting also can be examined under the adversarial or *iid* model but this section will operate under *iid*. This means that the impressions follow a distribution D where $P(i) = \frac{e_i}{n}$. Like the unweighted setting e_i is the expected number of times an impression i arrives and n is the total number of impressions that will arrive with $n = \sum_{i \in I} e_i$. Without loss of generality, it is assumed that $\forall_{i \in I}\ e_i = 1$. For any advertiser a, $\Gamma(a)$ is the set of edges touching a. The same definition is used for impressions. This setting also has identical definitions of "realization" graphs and competitive ratios as the unweighted setting. The goal then is to maximize the weights of edges assigned by the algorithm.

3.2 Algorithms and Theoretical Results

Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam developed new algorithms to handle the edge-weighted setting, however this section will focus on two particular algorithms [5]. Note that the authors did not name these algorithms so they will be referred to as HMZ 1 and HMZ 2. Consider the following LP model

based on the bipartite graph G given in the problem statement:

$$\max w^T p \text{ subject to:} \tag{3.1}$$

$$\forall a' \in A : \sum_{e \in \Gamma(a')} p_e \leq 1 \tag{3.2}$$

$$\forall i' \in I : \sum_{e \in \Gamma(i')} p_e \leq 1 \tag{3.3}$$

$$\forall e \in E : 0 \leq x_e \tag{3.4}$$

Algorithm 3: HMZ 1 [5].

1 **Offline Phase**:
2 Solve **LP** (3.1) and let p^* be an optimal solution vector.
3 **Online Phase**:
4 **for** $t = 1, \ldots, n$ **do**
5 Let an impression i' arrive at time t according to the probability distribution D.
6 Take edge $e \in \Gamma(i')$ with probability p_e^*.
7 Assign a to i' if a is available.

The analysis of HMZ 1 is simple. The probability that i arrives is $\frac{1}{n}$. The probability that for impression i the algorithm picks advertiser a is p_e. The probability that i does not arrive before time n is $\sum_{t=1}^{n-1}(1 - \frac{1}{n})^{t-1}$. Then the probability that any edge e, i is successfully matched is a is at least:

$$\sum_{t=1}^{n} \frac{p_e^*}{n}(1 - \frac{1}{n})^{t-1}$$

Using geometric series identities, the above expression can be simplified to:

$$\frac{p_e^*}{n} \frac{1 - (1 - \frac{1}{n})^n}{1 - (1 - \frac{1}{n})} = p_e^*(1 - (1 - \frac{1}{n})^n) \geq p_e^*(1 - \frac{1}{e})$$

With this analysis it can be concluded that HMZ 1 achieves an approximation factor of $1 - \frac{1}{e}$. HMZ 2 builds on HMZ 1 and improves the approximation factor even more. Consider the following LP model based on the bipartite graph G given in the problem statement:

$$\max w^T p \text{ subject to:} \tag{3.5}$$

$$\forall a' \in A : \sum_{e \in \Gamma(a')} p_e \leq 1 \tag{3.6}$$

$$\forall i' \in I : \sum_{e \in \Gamma(i')} p_e \leq 1 \tag{3.7}$$

$$\forall e' \in E : p_{e'} \leq 1 - \frac{1}{e} \tag{3.8}$$

$$\forall e \in E : 0 \leq x_e \tag{3.9}$$

It has been shown that the expected value of the matching HMZ 2 outputs is at least $(1 - \frac{1}{e})\omega(M_s) + 0.095\omega(M')$, where $\omega(M_s)$ is the sum of the weights of all edges in M_s. The same definition is used for $\omega(M')$. Using this lemma, the authors were able to prove that the expected value of HMZ 2 is at least $0.667 \cdot OPT$ where OPT is the solution to the **LP** (3.4) [5].

3.3 Experimental Design and Results

Two experiments were conducted to assess the performance of HMZ 1 and HMZ 2. In the first experiment $|A| = 100$ and $|I| = |T| = 500$ where T is the arrival sequence. The arrival sequence is randomly generated using a uniform distribution which means for every $i \in I, P(i) = \frac{1}{n}$ where $e_i = 1$ and $n = |T|$. Then for every $a \in A$, δ neighbors are chosen from $|I|$ where δ denotes the number of neighbors each advertiser has. For each neighbor chosen, an edge is created between the advertiser and the neighbor with a random weight between 0 and 1. Once the graph is finished, **LP** (3.1) and

Algorithm 4: HMZ 2 [5].

Offline Phase:
Solve **LP** (3.4) and let p^* be an optimal solution vector.
Compute matching M_s where the probability that edge e is in M_s is equal to p_e^*.
Compute the maximum weighted matching M_1 of graph G.
Let M' be $M_1 \setminus M_s$.
Online Phase:
for $t = 1, \ldots, n$ **do**
    Let an impression i' arrive at time t according to the probability distribution D.
    If this is the first time i' is arriving, assign it to it's matched node in M_s if such a node exists and is not taken.
    If this is the second time i' is arriving, assign it to it's matched node in M' if such a node exists and is not taken.
    Otherwise reject.

LP (3.4) are solved using a linear program. HMZ 1 and HMZ 2 are executed using a varying δ that takes values from $\{1, 2, 3, 4, 5, 6\}$. The competitive ratio is calculated using the performance of HMZ 1 and HMZ 2 and the optimal values calculated from the linear programs. The second experiment uses the exact same setting as the first except that instead of varying the δ, the $|T|$ arrivals are varied instead with δ is set equal to 3. The results of both experiments are listed below.

Table 3.1 HMZ 1 and HMZ 2 Algorithms on Varying δ with $|I| = 100, |J| = |T| = 500$

DELTA	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$
HMZ 1	0.55	0.65	0.59	0.71	0.66	0.64
HMZ 2	0.61	0.57	0.51	0.46	0.54	0.56

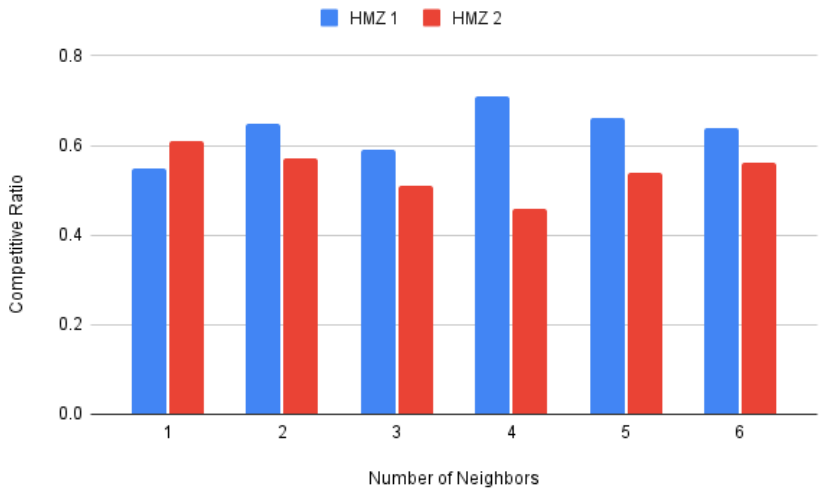


Figure 3.1 HMZ 1 and HMZ 2 Algorithms on Varying δ with $|U| = 100, |V| = |T| = 500$.

Table 3.2 HMZ 1 and HMZ 2 Algorithms on Varying $|T|$ with $|I| = |J| = 100$ and $\delta = 3$

ARRIVALS	$t = 50$	$t = 100$	$t = 300$	$t = 500$	$t = 800$	$t = 1000$
HMZ 1	0.40	0.74	0.88	0.90	0.93	0.86
HMZ 2	0.35	0.46	0.66	0.80	0.84	0.65

The performance of both algorithms are fairly good but what is interesting is how HMZ 1 is outperforming HMZ 2 despite HMZ 2 having higher theoretical bounds. It is not entirely clear why this is happening but one possible explanation is that HMZ 2 can only match every impression $i \in I$ at most twice. HMZ 1 can hypothetically match any impression more than two times. This means that if there are a lot of repeated impressions in the arrival sequence, HMZ 2 will simply reject them instead of trying to match them which can hurt performance. The only instance in which

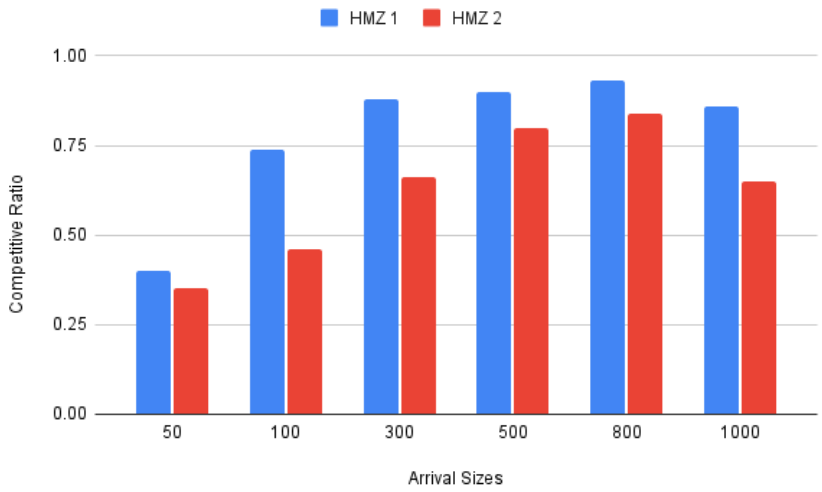


Figure 3.2 HMZ 1 and HMZ 2 Algorithms on Varying $|T|$ with $|I| = |J| = 100, \delta = 3$.

HMZ 2 outperforms HMZ 1 is the case where the number of neighbors equals 1. This behavior is puzzling and future investigation is needed to determine the reason.

CHAPTER 4

VERTEX WEIGHTED MATCHING

4.1 Problem Statement

Let $G(U, V, E)$ be a bipartite graph where U is the set of vertices each with a weight $b_u, u \in U$ and V is the set vertices that arrive in sequential order. While the weights of U are known the distribution of V is not known making this setting adversarial rather than the usual *iid* setting. As vertices from V arrive in sequential order, an irrevocable decision must be made. Each vertex $v \in V$ must be matched to a neighboring vertex $u \in U$ that has not been matched yet. This decision cannot be changed and if no such u exists then v will remain unmatched. Unmatched vertices cannot be matched later. The goal of any algorithm operating in this setting is the maximize the sum of vertex weights. This setting is very similar to edge-weighted online matching except that vertex weights are used rather than edge weights.

4.2 Algorithms and Theoretical Results

One of the first algorithms developed for vertex weighted matching was the Ranking algorithm created by Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani [8]. However, the ranking algorithm operates in a setting where $\forall_{u \in U}, b_u = 1$. In other words, it operates in an unweighted setting. The psuedo-code is given in algorithm 5. The following is a reworded proof from [1] that shows the Ranking algorithm achieves a competitive ratio of at least $(1 - \frac{1}{e})$ in expectation. Let $G(U, V, E)$ be a bipartite graph similar to the one described in the problem statement. Suppose $\forall_{u \in U}, b_u = 1$, that $|U| = |V| = n$, and G has a perfect matching. This means $OPT = n$ which is the optimal performance and the optimal matching is a perfect matching. Let $\sigma = (u_1, \ldots, u_n)$ be a permutation of U, where RANKING(σ) denotes the matching made the algorithm operating on permutation σ, and $\sigma(u) = t$ denotes the rank of

Algorithm 5: Ranking [1].

1 **Offline Phase**:
2 Choose a random permutation σ of U uniformly from the space of all permutations.
3 **Online Phase**:
4 **for** *each arriving* $v \in V$ **do**
5 Match v to the unmatched neighbor in u which appears earliest in σ.

vertex u. Then let:

$$y_{\sigma,i} = \begin{cases} 1 & \text{if the vertex at rank } i \text{ in } \sigma \text{ is matched by } RANKING(\sigma) \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Let Q_t be the set of all matched vertices:

$$Q_t = \{(\sigma, t) : y_{\sigma,t} = 1\} \tag{4.2}$$

Let R_t be the set of all unmatched vertices:

$$R_t = \{(\sigma, t) : y_{\sigma,t} = 0\} \tag{4.3}$$

Let $x_t = \frac{|Q_t|}{n!}$ and $1 - x_t = \frac{|R_t|}{n!}$. Then x_t represents the probability that the vertex at rank t is matched while $1 - x_t$ represents the probability that it is unmatched. It is easy to see that the expected value of the algorithm is $\mathbb{E}[ALG] = \sum_t x_t$. Next two definitions and one lemma must be established.

Definition 1 Let σ_u^i be the permutation where vertex u from permutation σ is moved and inserted into position i.

Definition 2 Let $f(\sigma, t)$ be the charging map which is a map from bad events to good events. Bad events are vertices that are unmatched while good evens are matched

vertices. More precisely:

$$f(\sigma,t) = \{(\sigma_u^i, s) : 1 \leq i \leq n,\ \sigma(u) = t,\ RANKING(\sigma_u^i)\ matches\ v^*\ to\ u'\ s.t.\ \sigma_u^i(u') = s\} \tag{4.4}$$

Lemma 1 If vertex u at rank t in σ is unmatched by $RANKING(\sigma)$, then for every $1 \leq i \leq n$, v^* is matching in $RANKING(\sigma_u^i)$ to a vertex u' s.t. $\sigma_u^i(u') \leq t$.

To simplify this statement, it is saying that for every unmatched vertex in σ there are other matched vertices in permutation σ_u^i. This lemma was proven in [1], and will come in handy for the proof. Since the charging map $f(\sigma,t)$ contains other permutations with good events, or matched vertices, it is obvious that $f(\sigma,t)$ is a subset of Q_t. Then for every $(\sigma,t) \in R_t$ and for every $s \leq t$:

$$\bigcup_{(\sigma,t)\in R_t} f(\sigma,t) \subseteq \bigcup_{s\leq t} Q_s \tag{4.5}$$

Now let κ and ω be two permutations. Suppose $(\sigma,s) \in f(\kappa,t)$ and $(\sigma,s) \in f(\omega,t)$. Then $(\sigma,s) = \kappa_u^i$ for some u and i. And $(\sigma,s) = \omega_u^i$ for some u and i. Then $\sigma = \kappa_u^i = \omega_u^i$. This implies $\kappa = \omega$. Which means that the charging map $f(\sigma,t)$ for a particular t and σ is unique. Thus:

$$1 - x_t = \frac{|R_t|}{n!} = \frac{1}{n} \cdot \frac{|\bigcup_{(\sigma,t)\in R_t} f(\sigma,t)|}{n!} \tag{4.6}$$

$$\frac{1}{n} \cdot \frac{|\bigcup_{s\leq t} Q_s|}{n!} = \frac{1}{n} \sum_{s\leq t} \frac{|Q_s|}{n!} = \frac{\sum_{s\leq t} x_s}{n} \tag{4.7}$$

Due to (4.5), it can be concluded that $1 - x_t \leq \frac{1}{n}\sum_{s\leq t} x_s$. Since the graph has a perfect matching and $OPT = n$, then $\frac{1}{n}\sum_{s\leq t} x_s$ is the competitive ratio. The value of this competitive ratio can be examined using recurrence relations. Let $S_t = \sum_{s\leq t} x_s$.

Then $1 - x_t \leq \frac{1}{n}\sum_{s\leq t} x_s$ can be expressed as $S_t(1+\frac{1}{n}) \geq 1 + S_{t-1}$ [2]. The base case is $S_0 = 0$. Then:

$$S_1 = \frac{1}{1+\frac{1}{n}} \tag{4.8}$$

$$S_2 = \frac{1+\frac{1}{1+\frac{1}{n}}}{1+\frac{1}{n}} = \frac{1}{1+\frac{1}{n}} + \frac{1}{(1+\frac{1}{n})^2} \tag{4.9}$$

$$S_3 = \frac{1+\frac{1}{1+\frac{1}{n}}+\frac{1}{(1+\frac{1}{n})^2}}{1+\frac{1}{n}} = \frac{1}{1+\frac{1}{n}} + \frac{1}{(1+\frac{1}{n})^2} + \frac{1}{(1+\frac{1}{n})^3} \tag{4.10}$$

This relation can be expressed as $S_t = \sum_{s=1}^{t} \frac{1}{(1+\frac{1}{n})^s}$. Then:

$$\sum_{s=1}^{t} \frac{1}{(1+\frac{1}{n})^s} = \sum_{s=1}^{t} \frac{1}{(\frac{n+1}{n})^s} = \sum_{s=1}^{t} \left(\frac{n}{n+1}\right)^s \tag{4.11}$$

$$\sum_{s=1}^{t} \left(\frac{n}{n+1}\right)^s = \sum_{s=1}^{t} \left(\frac{n+1}{n+1} - \frac{1}{n+1}\right)^s = \sum_{s=1}^{t} \left(1 - \frac{1}{n+1}\right)^s \tag{4.12}$$

Thus $\frac{1}{n}\sum_{s\leq t} x_s$ can be expressed as $\frac{1}{n}\sum_{s=1}^{t}\left(1-\frac{1}{n+1}\right)^s$. Using basic algebraic manipulation the infinite sum can be expressed as:

$$\frac{1}{n} \cdot \frac{1-\frac{1}{n+1}}{1-\frac{1}{n+1}} \cdot \sum_{s=1}^{t} \left(1 - \frac{1}{n+1}\right)^s = \frac{1-\frac{1}{n+1}}{n} \cdot \sum_{s=1}^{t} \left(1 - \frac{1}{n+1}\right)^{s-1} \tag{4.13}$$

Using geometric series where $a = 1$ and $r = (1 - \frac{1}{n+1})$, the expression evaluates to:

$$\frac{1-\frac{1}{n+1}}{n} \cdot \frac{1-\left(1-\frac{1}{n+1}\right)^n}{1-\left(1-\frac{1}{n+1}\right)} = \frac{\left(1-\frac{1}{n+1}\right)-\left(1-\frac{1}{n+1}\right)^{n+1}}{\frac{n}{n+1}} \tag{4.14}$$

This is equal to:

$$\frac{\left(1-\frac{1}{n+1}\right)-\left(1-\frac{1}{n+1}\right)^{n+1}}{1-\frac{1}{n+1}}=1-\left(1-\frac{1}{n+1}\right)^{n} \tag{4.15}$$

For the expression $1-\left(1-\frac{1}{n+1}\right)^{n}$, as n approaches infinity the expression approaches $1-\frac{1}{e}$. The Ranking achieves a high competitive ratio, the issue is that Ranking algorithm achieves this performance in a unweighted setting. For everyday e-commerce, vertices will usually take on weights. Thus the Perturbed-Greedy algorithm was proposed. The pseudo-code is detailed below where b_u is the weight for every vertex $u \in U$: As discussed in [1], the Perturbed-Greedy algorithm can be

Algorithm 6: Perturbed-Greedy [1].

Offline Phase:
For each $u \in U$, pick a number x_u uniformly at random from $[0, 1]$.
Define the function $\psi(x) = 1 - e^{-(1-x)}$
Online Phase:
for *each arriving* $v \in V$ **do**
    Match v to the unmatched neighbor in $u \in U$ with the highest value $b_u\psi(x_u)$. Break ties with vertex id.

thought as a non-uniform version of the Ranking algorithm. In the instance where each weight b_u is perturbed identically and independently, let the weights be arranged in decreasing order. If all weights are equal, then this turns into a random permutation of U. In other words, the Ranking algorithm is the Perturbed-Greedy algorithm in the case where all the weights are equal. In a very elegant proof that follows similar techniques to the proof for Ranking, the authors show that Perturbed-Greedy achieves a competitive ratio of $1-\frac{1}{e}$ and no other randomized algorithm can do better.

4.3 Experimental Design and Results

The experiments uses synthetic data which means it was generated by an algorithm. Two experiments are used, and in the first experiment the sets U, V, and T was set to be 100, so $|U| = |V| = |T| = 100$. U and V are two distinct sets belonging to the bipartite graph G, and T is the set of arrivals that arrive in the online phase. The parameter δ denotes the number of neighbors each $u \in U$ can have. This parameter was tested on six values $\{1, 2, 3, 4, 5, 6\}$. For each $u \in U$, δ neighbors are picked from V uniformly. Then a edge is constructed between u and every neighbor that was picked. An arrival sequence of size $|T|$ is generated uniformly from V. A linear program is then used to find the OPT which is detailed below where $a \sim b$ indicates b is a neighbor of a:

$$\max \sum_{u \in U} \sum_{u \sim v} b_u \cdot X_{uv} \quad \text{subject to:} \tag{4.16}$$

$$\forall u \in U : \sum_{u \sim v} X_{uv} \leq 1 \tag{4.17}$$

$$\forall v \in V : \sum_{v \sim u} X_{uv} \leq 1 \tag{4.18}$$

$$\forall e \in E : 0 \leq x_e \tag{4.19}$$

For the Ranking algorithm the weights b_u are set to 1 and for greedy-perturbed each weight is set to a random number between 0 and 10. Then the Ranking and Perturbed-Greedy algorithm is run on the generated arrival sequence and the competitive ratio is averaged over a 100 iterations. Both algorithms are run using their respective weights which means that the Ranking algorithm will use $b_u = 1$ while Greedy-Perturbed uses $b_u = (0, 10)$. The second experiment uses the same exact setting as the first experiment except for two differences. The first difference is that δ is fixed to 3. The second difference is the number of arrivals are varied using these values $\{50, 60, 70, 80, 90, 100\}$. The results are shown below in table 4.1, table 4.2, figure 4.1, and figure 4.2.

Table 4.1 Ranking and Greedy-Perturbed Algorithms on Varying δ with $|U| = |V| = |T| = 100$

DELTA	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$
RANKING	0.82	0.75	0.79	0.83	0.85	0.89
GREEDY-PERTURBED	0.63	0.74	0.80	0.90	0.89	0.93

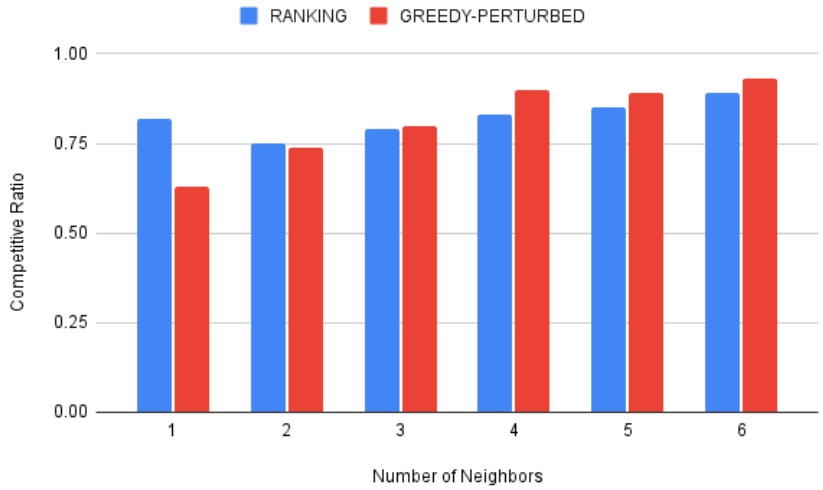


Figure 4.1 Ranking and Greedy-Perturbed on Varying δ with $|U| = |V| = |T| = 100$.

For varying δ, the performance between Ranking and Perturbed-Greedy is close to each other. As δ increases performance goes up, since the number of neighbors increases for each $u \in U$. This increases the odds that u gets matched since it has more options. The lowest competitive ratio achieved is 0.63 for $\delta = 1$ on Perturbed-Greedy. This is very close to the expected value for $\mathbb{E}[\frac{ALG}{OPT}] \geq 1 - \frac{1}{e}$ which was proven earlier.

For varying $|T|$, the performance between Ranking and Perturbed-Greedy is also close to each other. However, the performance does dip below the expected competitive ratio unlike the varying δ setting. For value $t = 50$ and $t = 60$ both

Table 4.2 Ranking and Greedy-Perturbed Algorithms on Varying $|T|$ with $|U| = |V| = 100$ and $\delta = 3$

ARRIVALS	$t = 50$	$t = 60$	$t = 70$	$t = 80$	$t = 90$	$t = 100$
RANKING	0.45	0.51	0.60	0.67	0.72	0.80
GREEDY-PERTURBED	0.54	0.56	0.74	0.70	0.78	0.79

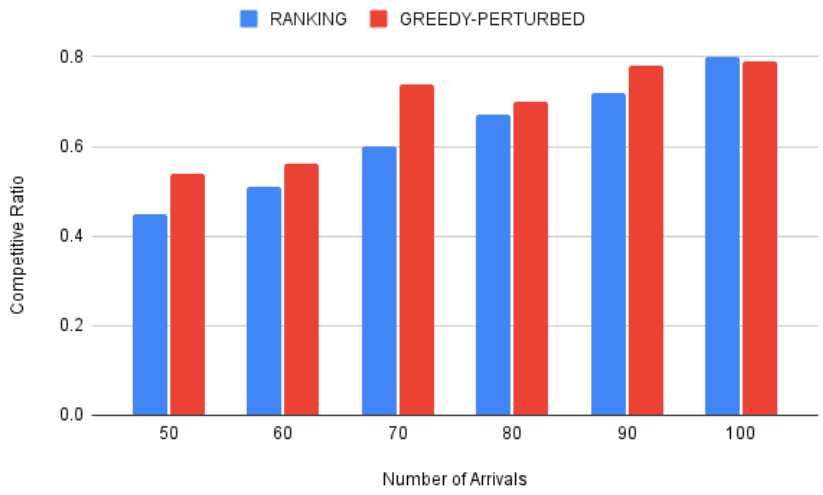


Figure 4.2 Ranking and Greedy-Perturbed on Varying $|T|$ with $|U| = |V| = 100$ and $\delta = 3$.

Ranking and Perturbed-Greedy both dip below the expected competitive ratio. For value $t = 70$, only Ranking dips below. Since these values of t are smaller than the size of $|U|$, it is unlikely that every $u \in U$ will be matched. Since there are unmatched $u \in U$, the competitive ratio takes a hit. As t increases, it can be observed that the competitive ratio increases and behaves more accordingly to the proven theoretical bounds.

CHAPTER 5

FAIRNESS MAXIMIZATION

5.1 Problem Statement

The previous sections have been focusing on hypothetical settings without any applications to real world problems. This section will focus on using online algorithms for maximizing fairness for real world applications. It is important to note that online matching markets (OMMs) have gained popularity over the past couple of decades with notable examples such as Uber, Google Advertising, and Amazon Mechanical Turks. Online agents arrive dynamically and the goal is assign each online agent to an offline agent such that profit is maximized. In the case of ridesharing platforms, like Uber and Lyft, the offline agents would be the drivers and the online agents would be the app users. Unfortunately, OMMs have been dealing with bias and discrimination that ultimately lowers fairness. For example, in ridesharing platforms it has been reported that black passengers have to wait on average 1 minute and 43 seconds longer to be matched and that drivers are 4% more likely to cancel on them [6]. As a result, researchers have been studying this area in an attempt to increase fairness without compromising profits.

The setting for fairness maximization is as follows: Let $G(I, J, E)$ be a bipartite graph, where I represents the set of offline agents, J represents the set of online agents, and E represents the set of edges between I and J. Let T be the number of rounds during the online phase. During T, arrivals are sampled from J with replacement. This forms a probability distribution D where $P(j \text{ arrives}) = \frac{r_j}{|T|}$, where r_j is the number of times j was sampled in T. Clearly $\sum_{j \in J} \frac{x_j}{|T|} = 1$. As the online agents arrive according to the distribution T, an irrevocable and irreversible decision must be made which is to either match the online agent to an offline neighbor in I or to reject it. Two assumptions are made in this sections, the first is that the setting

operates under KIID or known, independent, and identical distribution. The second assumption is that each offline agent can only be matched once where $Z_i = 1$ indicates that offline agent i is matched. The goal is to maximize fairness which is defined as $\min_{i \in I} \mathbb{E}[Z_i]$. This is also known as individual fairness maximization or IFM. This section will also examine fairness maximization under VOM which shares the same setting as IFM except that the weights for each vertex are weighted.

5.2 Algorithms and Theoretical Results

The algorithms in this section uses optimal value calculated from the following benchmark linear program as a guide. Just like in the previous sections $i \sim j$ indicates i is a neighbor of j and $j \sim i$ indicates j is a neighbor of i. Let N_i be the set of neighbors of i, where N_j is defined similarly. x_{ij} is the expected number of times the edge (i, j) is matched by the linear program [9].

$$\text{IFM:} \quad \max \min_{i \in I} \sum_{j \sim i} x_{ij} \tag{5.1}$$

$$\text{VOM:} \quad \max \sum_{i \in I} w_i \cdot \sum_{j \sim i} x_{ij} \quad \text{subject to:} \tag{5.2}$$

$$\forall j \in J : \sum_{i \sim j} x_{ij} \leq 1 \tag{5.3}$$

$$\forall i \in I : \sum_{j \sim i} X_{ij} \leq 1 \tag{5.4}$$

$$\forall i \in I, \forall S \subseteq N_i, |S| = O(1) : 0 \leq \sum_{j \in S} x_{ij} \leq 1 - e^{-|S|} \tag{5.5}$$

$$\forall e \in E : 0 \leq x_e \tag{5.6}$$

The first algorithm that this section examines is the SAMP-B algorithm from [9]. SAMP-B is a fairly straightforward algorithm that samples offline agents i using a probability distribution that is constructed from the optimal solution $\{x_{ij}\}$. The authors were able to show that SAMP-B achieves a competitive ratio of at least 0.725 on IFM which beats the $1 - \frac{1}{e}$ barrier. However, they also show that under the

Algorithm 7: Sampling with Boosting (SAMP-B).

Offline Phase:
Solve LP (5.1) and let $\{x_{ij}\}$ be an optimal solution.
Online Phase:
for $t = 1, \ldots, T$ **do**
    Let online agent of type j arrive at the beginning of time t.
    Let $N_{j,t} = \{i \in N_j, i \text{ is available at } t\}$.
    if $N_{j,t} = \emptyset$ **then**
        Reject j.
    else
        Sample a neighbor $i \in N_{j,t}$ with probability $\frac{x_{i'j}}{\sum_{i' \in N_{j,t}} x_{i',j}}$.

VOM setting SAMP-B can never break the $1 - \frac{1}{e}$ barrier without attenuation. Thus SAMP-AB was created [9] and it has been demonstrated that SAMP-AB achieves a competitive ratio of at least 0.719 for VOM. The pseudo-code for SAMP-AB is in the experimental section due to space constraints.

5.3 Experimental Design and Results

The experimental setting for IFM is as follows. A bipartite graph $G(I, J, E)$ is constructed where $|I| = 100$, $|J| = |T|$. For each offline agent $i \in I$, δ random neighbors are chosen from J. An edge is constructed between each offline agent i and each of it's neighbors. Integral arrival rates are assumed which means $r_j = 1$ for every j. Then the set of arrivals T are uniformly generated since $r_j = 1$. Then SAMP-B is executed on T and the performance is compared against other algorithms. Four other algorithms are used mainly Greedy, Ranking, BSSX, and MGS. The Greedy algorithm is simple and assigns to each online agent j at arrival an available neighbor where ties are broken randomly. For the Ranking algorithm, it assigns j an available neighbor according to a uniform and random permutation that is created during the offline phase. The details of BSSX and MGS will be left out and instead will be

Algorithm 8: Sampling with Attenuation and Boosting (SAMP-AB).

Offline Phase:
Solve LP (5.1) and let $\{x_{ij}\}$ be an optimal solution.
Initialization: Let $\beta_{i,t} = 1$ for $t = 1$ and for all $i \in I$.
for $t = 2, 3, \ldots, T$ **do**
    Use the Monte-Carlo method to simulate step 10 to step 16 for all rounds $t' = 1, 2, \ldots, t-1$ of the online phase to calculate the probability $\alpha_{i,t}$ that each offline agent i is active at the beginning t.
    Set $\beta_{i,t} = \min\left(1, \frac{(1-\frac{1}{T})^{t-1}}{\alpha_{i,t}}\right)$.
Online Phase:
Initialization: Label all offline vertices active at $t = 1$.
for $t = 1, \ldots, T$ **do**
    Independently relabel each active offline agent i as active and inactive with respective probabilities $\beta_{i,t}$ and $1 - \beta_{i,t}$.
    Let online agent of type j arrive at the beginning of time t.
    Let $N_{j,t} = \{i \in N_j, i \text{ is available at } t\}$.
    if $N_{j,t} = \emptyset$ **then**
        Reject j.
    else
        Sample a neighbor $i \in N_{j,t}$ with probability $\frac{x_{i'j}}{\sum_{i' \in N_{j,t}} x_{i',j}}$.

referenced in [3] and [10]. respectively. Four experiments will be conducted where in the first experiment the number of neighbors δ will be varied using values from $\{2, 3, 4, 5, 6\}$ and $|I| = |J| = |T| = 100$. In the second experiment, the number of neighbors will be varied just like in the first experiment except that $|I| = 100$ and $|J| = |T| = 500$. In the third experiment the number of arrivals T will be varied using values from $\{50, 100, 300, 500, 1000\}$ with $\delta = 2$. In the last experiment, the number of arrivals will be varied using the same values from the third experiment except $\delta = 3$. The experimental setting for VOM is almost identical to IFM except for three differences. The first difference is two competitive ratios are used CR1 and CR2 which

are defined as $\min_i \frac{\mathbb{E}[Z_i]}{x_i^*}$ and $\sum_i w_i \cdot \mathbb{E}[Z_i]$ respectively. The second difference is that two experiments are conducted instead of four. For varying δ the values are varied from $\{2, 3, 4, 5, 6\}$ with $|I| = |J| = |T| = 100$ for both competitive ratios. For varying T the values are varied from $\{70, 80, 90, 100\}$ with $\delta = 3$ and $|I| = |J| = |T|$ for both competitive ratios. The last difference is that two new algorithms are added into the experiments: SAMP-AB and JL from [7]. The results are shown below.

Table 5.1 Varying δ on IFM with fixed T, $|I| = 100, |J| = |T| = 100$

DELTA	SAMP-B	GREEDY	RANKING	BSSX	MGS
$\delta = 2$	0.78	0.30	0.37	0.62	0.48
$\delta = 3$	0.76	0.29	0.42	0.43	0.47
$\delta = 4$	0.75	0.38	0.44	0.55	0.46
$\delta = 5$	0.74	0.50	0.50	0.53	0.47
$\delta = 6$	0.78	0.52	0.48	0.45	0.56

Table 5.2 Varying δ on IFM with fixed T, $|I| = 100, |J| = |T| = 500$

DELTA	SAMP-B	GREEDY	RANKING	BSSX	MGS
$\delta = 2$	0.78	0.51	0.57	0.68	0.42
$\delta = 3$	0.75	0.58	0.59	0.70	0.50
$\delta = 4$	0.79	0.80	0.73	0.82	0.65
$\delta = 5$	0.91	0.90	0.89	0.88	0.62
$\delta = 6$	0.92	0.90	0.88	0.89	0.70

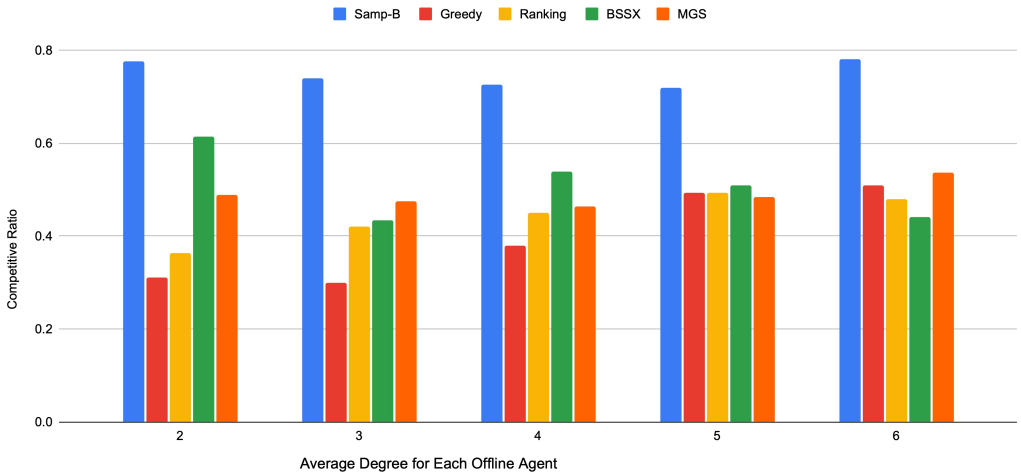


Figure 5.1 Varying δ on IFM with fixed T, $|I| = 100, |J| = |T| = 100$.

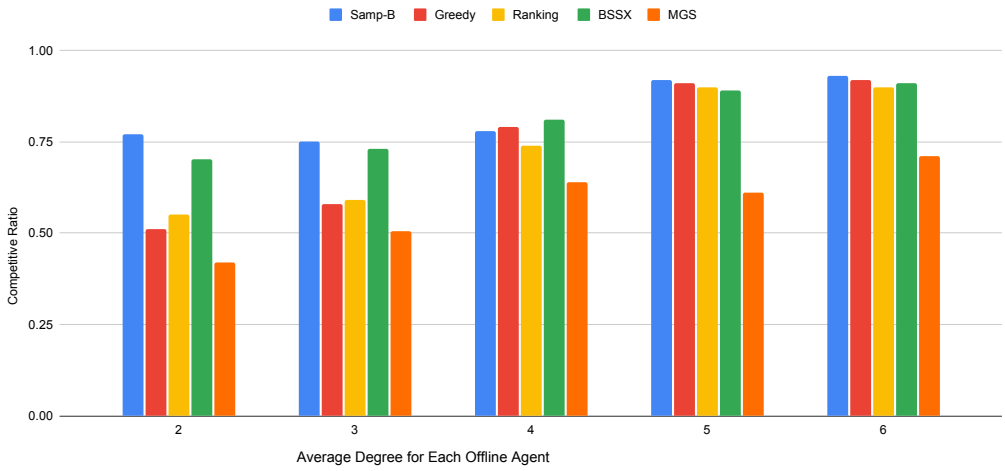


Figure 5.2 Varying δ on IFM with fixed T, $|I| = 100, |J| = |T| = 500$.

Table 5.3 Varying T on IFM with fixed $\delta = 2$, $|I| = 100, |J| = |T|$

T	SAMP-B	GREEDY	RANKING	BSSX	MGS
$T = 50$	0.70	0.30	0.27	0.48	0.39
$T = 100$	0.71	0.29	0.22	0.37	0.32
$T = 300$	0.73	0.54	0.52	0.58	0.42
$T = 500$	0.70	0.68	0.54	0.60	0.57
$T = 1000$	0.71	0.72	0.61	0.74	0.58

Table 5.4 Varying T on IFM with fixed $\delta = 3$, $|I| = 100, |J| = |T|$

T	SAMP-B	GREEDY	RANKING	BSSX	MGS
$T = 50$	0.71	0.21	0.28	0.32	0.31
$T = 100$	0.78	0.38	0.41	0.46	0.24
$T = 300$	0.72	0.61	0.53	0.63	0.42
$T = 500$	0.77	0.60	0.61	0.69	0.53
$T = 1000$	0.82	0.83	0.84	0.82	0.65

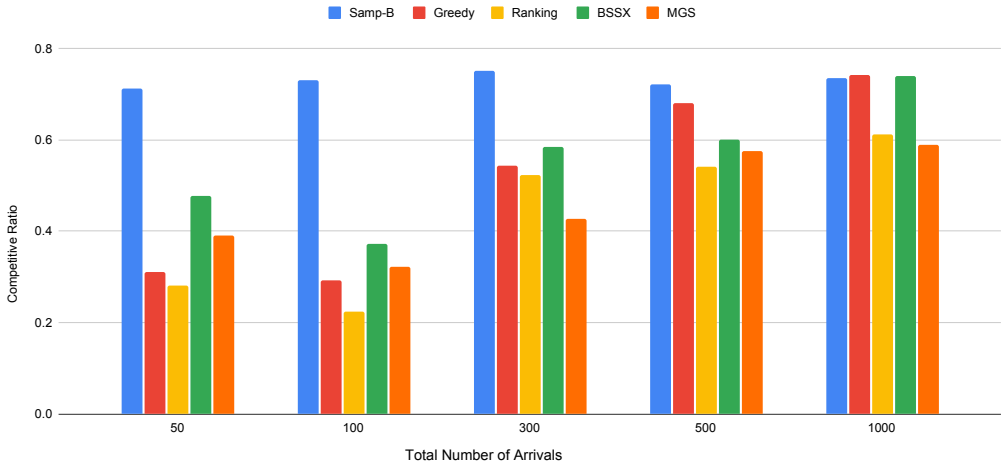


Figure 5.3 Varying T on IFM with fixed $\delta = 2$, $|I| = 100, |J| = |T|$.

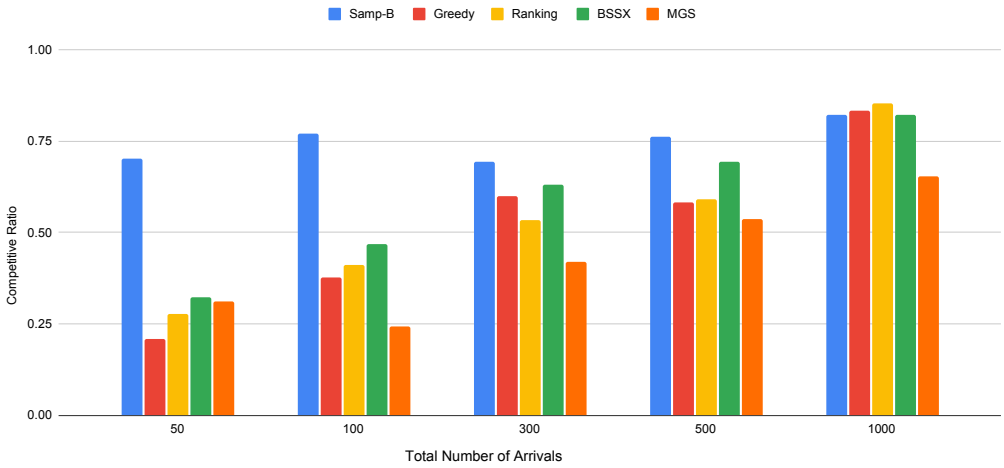


Figure 5.4 Varying T on IFM with fixed $\delta = 3$, $|I| = 100, |J| = |T|$.

Table 5.5 Varying δ on VOM CR1, $|I| = |J| = |T| = 100$

DELTA	SAMP-B	SAMP-AB	GREEDY	RANKING	JL	BSSX	MGS
$\delta = 2$	0.62	0.43	0.40	0.16	0.49	0.31	0.39
$\delta = 3$	0.69	0.61	0.52	0.22	0.48	0.42	0.35
$\delta = 4$	0.75	0.68	0.63	0.14	0.40	0.42	0.46
$\delta = 5$	0.77	0.76	0.73	0.41	0.39	0.45	0.50

Table 5.6 Varying δ on VOM CR2, $|I| = |J| = |T| = 100$

DELTA	SAMP-B	SAMP-AB	GREEDY	RANKING	JL	BSSX	MGS
$\delta = 2$	0.82	0.64	0.85	0.64	0.69	0.73	0.68
$\delta = 3$	0.83	0.72	0.84	0.65	0.71	0.75	0.66
$\delta = 4$	0.80	0.73	0.83	0.65	0.73	0.81	0.67
$\delta = 5$	0.83	0.78	0.86	0.65	0.72	0.84	0.68

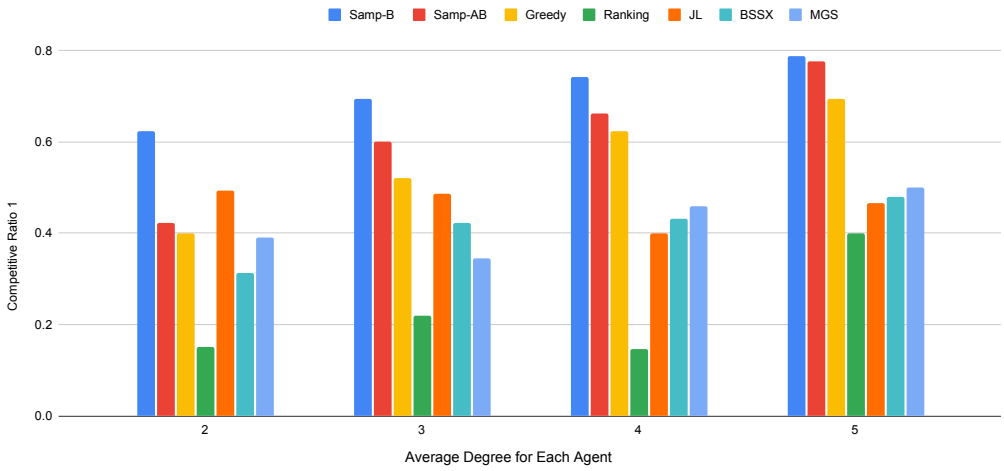


Figure 5.5 Varying δ on VOM CR1, $|I| = |J| = |T| = 100$.

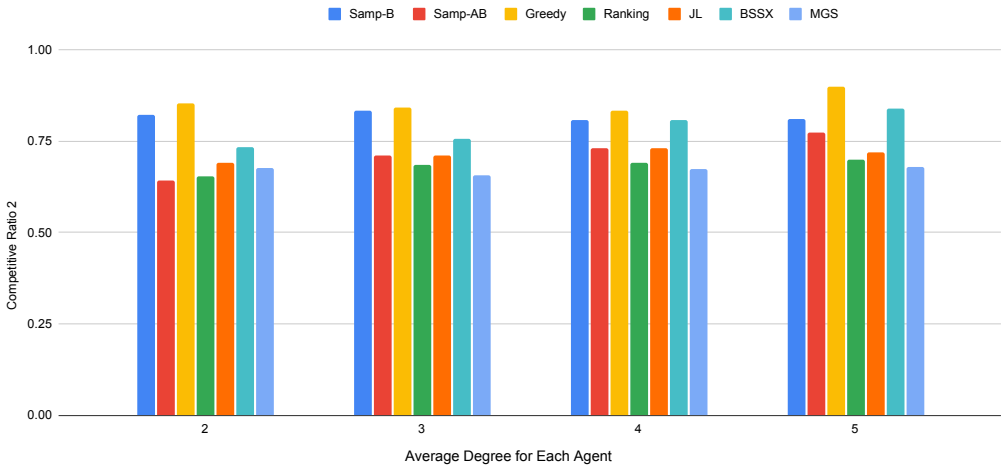


Figure 5.6 Varying δ on VOM CR2, $|I| = |J| = |T| = 100$.

Table 5.7 Varying T on VOM CR1 with $\delta = 3$, $|I| = |J| = |T|$

T	SAMP-B	SAMP-AB	GREEDY	RANKING	JL	BSSX	MGS
$t = 70$	0.68	0.60	0.49	0.06	0.54	0.47	0.38
$t = 80$	0.64	0.58	0.54	0.18	0.48	0.42	0.36
$t = 90$	0.67	0.65	0.58	0.28	0.45	0.48	0.38
$t = 100$	0.66	0.60	0.56	0.19	0.35	0.54	0.37

Table 5.8 Varying T on VOM CR2 with $\delta = 3$, $|I| = |J| = |T|$

T	SAMP-B	SAMP-AB	GREEDY	RANKING	JL	BSSX	MGS
$t = 70$	0.83	0.72	0.88	0.69	0.78	0.73	0.65
$t = 80$	0.84	0.73	0.87	0.68	0.76	0.79	0.63
$t = 90$	0.81	0.71	0.85	0.64	0.77	0.77	0.65
$t = 100$	0.82	0.73	0.86	0.66	0.78	0.74	0.66

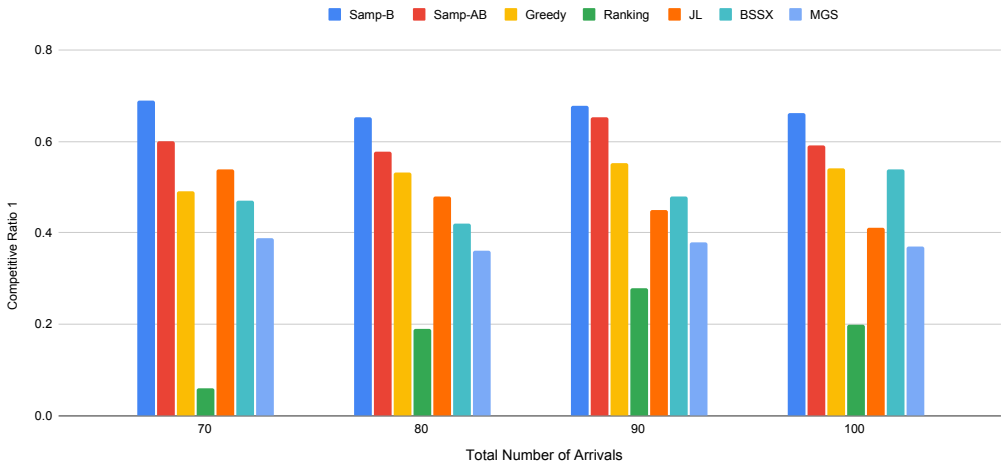


Figure 5.7 Varying T on VOM CR1 with $\delta = 3$, $|I| = |J| = |T|$.

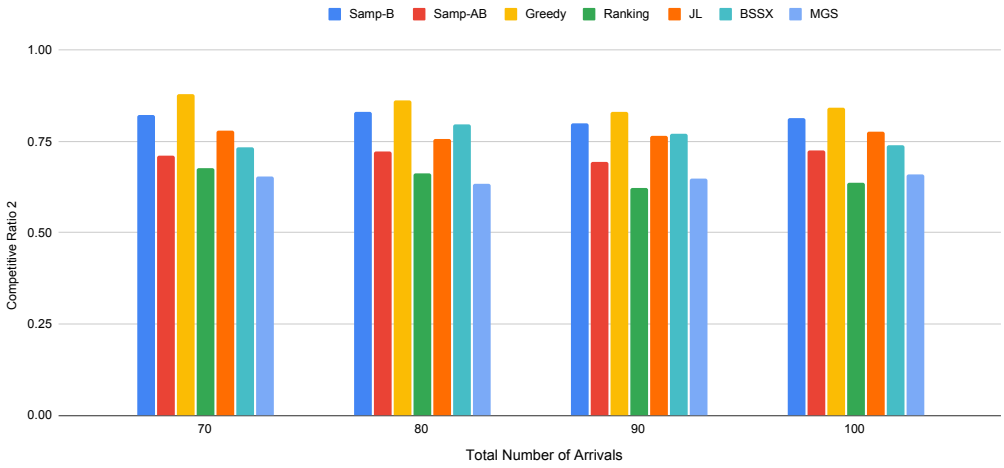


Figure 5.8 Varying T on VOM CR2 with $\delta = 3$, $|I| = |J| = |T|$.

For Table 5.1 and Figure 5.1, SAMP-B performed the best maintaining a competitive ratio of at least 0.74. Ranking, BSSX, and MGS were all around the same performance and greedy performed the worst with ranking coming in as a close second. As δ kept increasing Greedy, Ranking, BSSX, and MGS started having similar performance. This is because with a larger δ it becomes easier to increase fairness

due to the larger number of neighbors each j has. The most likely explanation as to why greedy performed so bad is that greedy simply assigns any online agent any available offline agent. This strategy is very random and does not make use of the optimal solution that was computed by the linear program. This could also explain why ranking performs similarly to greedy since the ranking algorithm picks a random permutation instead of using the optimal solution.

For Table 5.2 and Figure 5.2 performance is similar to before when $\delta = 2$. However as δ increases, the performance of all algorithms increases because each algorithm has more neighbors to work with which makes it easier to optimize fairness. Interestingly, the MGS algorithm performs fairly poor. One explanation is the MGS algorithm makes use of two matchings which means when a request of type j appears more than two times it gets rejected. Since The number of $|T|$ has increased this can increase the chance of repeats which can cause the MGS algorithm to take a hit.

For Table 5.3 and Figure 5.3, SAMP-B is the clear winner however as $|T|$ increases the Greedy algorithm and BSSX start approaching the performance of SAMP-B. The MGS and Ranking algorithm trail behind the others. MGS is most likely still suffering from the fact that it cannot assign any online agent j more than once. The Ranking algorithm is deterministic because it acts accordingly to a permutation that does not adapt or change which might be why it suffers in this case.

For Table 5.4 and Figure 5.4 the performance follows similar trends to Table 5.3 and Figure 5.3, except for one thing. The ranking algorithm performs a lot better. Although the ranking algorithm is still a deterministic algorithm, the increase in the number of neighbors from 2 to 3 means it will have more variety in it's decision making.

For Tables 5.5, 5.6 and Figures 5.5, 5.6, SAMP-B and SAMP-AB perform the best on CR1 which is the competitive ratio that measures fairness. For small δ, SAMP-B performs the best while the other algorithms trail behind. As δ increases,

SAMP-AB and Greedy increase in performance and get close to the performance of SAMP-B. The other four algorithms trail behind. For CR2 which measures maximizing weight values, all algorithms perform well with SAMP-B and Greedy having the best performance. Not surprisingly Greedy outperforms SAMP-B for CR2 since the Greedy algorithm always picks the vertex with the largest weight.

For Tables 5.7, 5.8 and Figures 5.7, 5.8, in the case of CR1, SAMP-B is the winner. But this is also the first time it performs below the theoretical lower bound of 0.725. The low number of neighbors and low $|T|$ sizes, makes it difficult for all algorithms to optimize for fairness and as a result all algorithms take a hit in performance. SAMP-AB is also clearly the second best performer with the ranking performance having the lowest performance. For CR2, SAMP-B and Greedy are the winners just like before. Greedy also outperforms SAMP-B for CR2.

CHAPTER 6

COMPARISON

6.1 Comparing Problem Statements

While problem statements between each section differ, there are a lot more similarities between each section than differences. For instance, in edge-weighted online matching the setting uses a bipartite graph $G(A, I, E)$ where A is the set of advertisers, I is the set of impressions and E is the set of edges. Impressions arrive in sequential order and the goal of the algorithm is to match each impression to an advertiser such that the weights of all edge matches is maximized. Additionally, it operates under *iid* which means the distribution of I is known beforehand. This is identical to the unweighted edge matching, the only difference is that the edges are not weighted. This means unweighted edge matching can be modeled as a special case of edge weighted matching where all weights $w = 1$. Vertex weighted matching is also fairly similar to edge weighted matching and unweighted edge matching. The main difference is instead of using edge weights it uses the weights of each vertex. VOM is also the only problem setting where it operated in the adversarial setting which means the algorithms know nothing about the distribution of V. Additionally instead of using advertisers A and impressions I the setting used vertices U and V. However this is just a difference in naming, overall the setting is very similar for the most part. IFM is the most different one. Rather than maximizing the sum of some weights, IFM focused on maximizing fairness which is defined as $\max \min_{i \in I} \mathbb{E}[Z_i]$. Besides the objective function, IFM also uses bipartite graphs where the goal is the assign incoming arrivals $j \in J$ to an available neighbor $i \in I$. IFM also operates under the non-adversarial setting which is known as $KIID$ instead of *iid*.

6.2 Comparing Algorithms and Theoretical Results

For unweighted edge matching, Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan introduce simple algorithms that are designed to achieve a competitive ratio of at least $1 - \frac{1}{e}$. Then they introduce more complicated algorithms designed to break the $1 - \frac{1}{e}$ barrier. In weighted edge matching, Bernhard Haeupler, Vahab Mirrokni, and Morteza Zadimoghaddam follow the same procedure. The TSM algorithm is supposed to achieve a ratio of at least 0.67 while HMZ 2 is supposed to achieve a ratio of at least 0.667. What is interesting is that both algorithms make use of a two matchings system where if any impression i arrives more than 2 times then it should be rejected. Both algorithms also under perform in certain settings to their simpler counterparts which are the SM algorithm and HMZ 1. They might share weaknesses since they both use two matchings, such as being unable to deal with a large number of repeats. However, the TSM algorithm never uses the optimal solution calculated by the linear program and instead uses max flow instead. HMZ 2 differs in that it does.

The VOM setting is interesting in that the authors show that the $1 - \frac{1}{e}$ barrier can never be broken in the weighted setting. So rather than improving the competitive ratio they simply introduce an algorithm that can work in a weighted setting and achieve a ratio of $1 - \frac{1}{e}$. The author also introduces a simple algorithm as a warm up then moves on to the more complicated algorithm later. This seems to be a fairly common practice.

SAMP-B is shown to have a competitive ratio of 0.725 and the ratio of SAMP-AB is 0.719 which is a bigger leap in performance when compared to the algorithms from other sections. SAMP-B and SAMP-AB use a probabilistic approach rather than a deterministic approach like TSM or Ranking. SAMP-B, SAMP-AB, Ranking, perturbed-greedy also seem the most robust and perform closer to their theoretical bounds. On the other hand, the performance for TSM, SM, HMZ 1,

and HMZ 2 have a lot more variation in performance and will under perform their theoretical bounds in specific settings.

Out of the 8 algorithms that were surveyed, SAMP-B and SAMP-AB performed the best and are the most robust. These algorithms are more oriented towards practical applications, while the algorithms that were more oriented towards theoretical applications were not as robust. The other 6 algorithms varied more in performance. In general, it is better to use algorithms that are designed towards real world applications since these type of algorithms are usually extensively tested in experiments. One area this survey struggled in is experimental design. Most of the papers that were surveyed did not include any experiments. The papers that did include experiments did not give much justification in how they set up their experiments.

BIBLIOGRAPHY

[1] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 07 2010.

[2] Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39:80–87, 03 2008.

[3] Brian Brubach, Karthik Sankararaman, Aravind Srinivasan, and Pan Xu. Attenuate locally, win globally: Attenuation-based frameworks for online stochastic matching with timeouts. *Algorithmica*, 82, 01 2020.

[4] Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 117–126, 2009.

[5] Bernhard Haeupler, Vahab Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. pages 170–181, 12 2011.

[6] Kristin Houser. Uber and lyft still allow racist behavior, but not as much as taxi services, 2018. Accessed on March, 20, 2023.

[7] Patrick Jaillet and Xin Lu. Online stochastic matching: New algorithms with better bounds. *Mathematics of Operations Research*, 39, 04 2012.

[8] Richard M. Karl, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC 1990)*, pages 352–358, 1990.

[9] Will Ma, Pan Xu, and Yifan Xu. Fairness maximization among offline agents in online-matching markets. *17th Conference on Web and Internet Economics*, 2021.

[10] Vahideh H. Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *CoRR*, abs/1007.1673, 2010.