ALLOCATION PROBLEMS WITH PARTIAL INFORMATION

A Thesis
Presented to
The Academic Faculty

by

Pushkar Tripathi

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Algorithms, Combinatorics, and Optimization

College of Computing
Georgia Institute of Technology
August 2012

ALLOCATION PROBLEMS WITH PARTIAL INFORMATION

Approved by:

Professor Vijay Vazirani, Advisor
College of Computing
Georgia Institute of Technology

Professor Ozlem Ergun
Industrial and Systems Engineering
Georgia Institute of Technology

Professor Prasad Tetali
School of Mathematics
Georgia Institute of Technology

Professor Maira Florina Balcan
College of Computing
Georgia Institute of Technology

Professor Shabbir Ahmed
Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: 25 May 2012

To my father,

Dr. B.M. Tripathi,

for inspiring me to follow my dreams.

ACKNOWLEDGEMENTS

I would like to thank my advisor Vijay Vazirani, for his constant support and motivation throughout my graduate studies and for being so patient with me. His wisdom and insight were instrumental in the successful completion of this thesis. I was fortunate to work with Prasad Tetali towards the end of my doctoral studies and I benefited immensely from his unique perspective and expertise.

I greatly appreciate the guidance afforded by Gagan Goel and Aranyak Mehta as mentors during my internships, and also as friends. I will always cherish the time I spent interacting with them and would continue to look up to them for inspiration and advice. I learnt a lot from them, ranging from research to writing and presenting and owe a lot of my research to the problems that they introduced me to.

I was blessed with plenty of friends at Georgia Tech, both within and outside the theory group who made the last four years very memorable. I cannot thank enough, my sister Samidha for being a pillar of strength and for picking up the slack when we were faced with hard times. Finally, I am grateful to my mother for her support and encouragement. She always urged me strive for excellence and had faith in me even when I was uncertain of my abilities.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Allocation problems have been central to the development of the theory of algorithms and also find applications in several realms of computer science and economics. In this thesis we initiate a systematic study of these problems in situations with limited information.

Towards this end we explore several modes by which data may be obfuscated from the algorithm. We begin by investigating temporal constraints where data is revealed to the algorithm over time. Concretely, we consider the online bipartite matching problem in the unknown distribution model and present the first algorithm that breaches the $1 - 1/e$ barrier for this problem.

Next we study issues arising from data acquisition costs that are prevalent in ad-systems and kidney exchanges. Motivated by these constraints we introduce the query-commit model and present constant factor algorithms for the maximum matching and the adwords problem in this model.

Finally we assess the approximability of several classical allocation problems with multiple agents having complex non-linear cost functions. This presents an additional obstacle since the support for the cost functions may be extremely large entailing oracle access. We show tight information theoretic lower bounds for the general class of submodular functions and also extend these results to get lower bounds for a subclass of succinctly representable non-linear cost functions.

CHAPTER I

INTRODUCTION

Allocation problems have been intensely studied by the computer science and operations research community over the last three decades and have been pivotal in the development of the theoretical underpinnings of these fields. Apart from their mathematical elegance these problems are also important from a practical viewpoint as they arise in a variety of areas ranging from job scheduling to bandwidth allocation. Furthermore, due to the explosive growth of the Internet these problems have also taken center stage in several realms of electronic commerce such as ad-auctions and display advertising.

A common characteristic of real world instantiations of these problems is the lack of complete and accurate information and as a result traditional models of optimization are often found wanting in such situations. Modeling and addressing this challenge is the central theme of this thesis. Concretely, in this thesis we will investigate this issue along the following three axes.

1. **Temporal Data:** These constraints arise when data is revealed to us over time; whereas at every stage the algorithm is expected to make irrevocable decisions based on the available information. For example, in ad-serving queries/requests come online throughout the day and need to be processed as they arrive. Clearly, we cannot afford to wait for all the data to be available and are therefore required to make decisions on the fly.

2. **Data Acquisition Costs:** In some problems the cost of acquiring data can also dictate our design choices. For example, in kidney exchanges the cost of medical testing for compatibility is prohibitively large; therefore we must be

judicious in choosing the tests to be performed. In such situations we cannot a priori assume access to all the information and need to design cost effective methods to acquire only the most relevant information to make an informed decision.

3. **Non-linear Objective Functions:** A major focus of computer science has been on the study of linear cost functions owing to their simplicity and malleability. However linear cost functions do not always capture the intricate dependencies that exist in real-world settings. For example the marginal (incremental) cost of producing a good only decreases with increasing scale of production. This property cannot be modeled though linear cost functions. Therefore linear functions only serve as an approximation to the original functions, thus even though we may have a good approximation algorithm for solving the linear optimization problem the output is still suboptimal. In this line of research we seek to model non-linear objectives and assess the limits of approximability of classical optimization problems in this paradigm.

1.1 Results and Organization

In this thesis we will address each of the issues mentioned above. Chapter 2 deals with temporal constraints while Chapters 3, 4 and 5 tackle problems where data acquisition presents a bottleneck. Finally in Chapter 6 we assess approximability of allocation problems with non-linear objective functions. A brief summary of our results is given below.

1.1.1 Temporal Data

In this domain we consider the online bipartite matching problem first studied by Karp et. al. in [37]. In this problem we are given one of the sides of a bipartite graph while the vertices from the other side arrive online, one at a time. For each

incoming vertex we are revealed its neighbors in the given side and it needs to be matched irrevocably upon arrival (whenever possible). The objective is to maximize the number of vertices that get matched.

Our main result deals with the version of this problem when the arriving vertices are drawn from a fixed but unknown distribution that may have an exponentially large support. For this problem we present a randomized algorithm that attains a competitive ratio of 0.656. In fact our results hold in the more general *random order arrival* model (incoming vertices arrive in random order). We also show an example to establish that our analysis was off by a factor of at most 0.03. The above results are based on joint work with Aranyak Mehta and Chinmay Karande [36].

1.1.2 Data Acquisition Costs

In order to model issues that arise due to data acquisition costs we define the query commit model. In this model we are given the vertex set of a general graph. For every pair of vertices we are *not* told a priori whether there is an edge connecting them, until we *probe/scan* this pair. If we scan a pair of vertices and find that there is an edge connecting them we are constrained to *pick* this edge and in this case both the vertices are removed from the graph. However, if we find them to be not adjacent, they continue to be available to be matched in the future. The goal is to maximize the number of vertices that get matched.

We present a randomized algorithm for this problem that attains a factor of 0.56. Our algorithm is a significant improvement on a 15 year old result by Aronson, Dyer, Frieze, and Suen in [2] where they analyzed a closely related randomized algorithm and showed that it attained a factor of 0.50000025. From the hardness perspective we consider a large family of algorithms (called vertex iterative algorithms), that iterate through the vertices and explore their neighborhood and show that no vertex iterative algorithm can attain a factor better than 0.75 for this problem.

We also study a stochastic relaxation of this model where we assume that for every pair of vertices we are additionally given the probability that they are adjacent. For this we give a 0.571 factor algorithm using a novel sampling technique that might be of interest in other domains of approximation algorithms. We also showed that no randomized algorithm could achieve a competitive ratio greater than 0.896.

Finally we consider the generalized assignment problem (GAP) within the query commit framework. GAP is a weighted generalization of the matching problem but unlike the max. matching problem it is known to be NP-hard even in the complete information setting. Using the sampling technique alluded to above we develop a $1 - 1/e$ factor algorithm for this problem.

The results in this section are based on joint work with Gagan Goel [25] and Kevin Costello and Prasad Tetali [12].

1.1.3 Non-linear Objective Functions

In the last part of this thesis we explore assignment problems with non-linear objective functions. In this framework multiple agents wish to collaborate to build a large combinatorial structure over a given graph whilst minimizing the total cost. Motivated by properties such as decreasing marginal cost and economies of scale, we use submodular functions to model the agents' cost functions. Even though this model generalizes the classical paradigm of linear cost functions these problems are not amenable to analysis by traditional techniques of combinatorial optimization.

We study the limits of approximability of the following classical problems in this paradigm - combinatorial reverse auctions and minimum cost perfect matching - and present (tight) polynomial information theoretic lower bounds. We observe that the polynomial information theoretic lower-bounds indicate that it is the lack of information and not the combinatorial structure that is impeding the search for efficient algorithms. To rectify this we also investigate the approximability of these problems

in a succinctly representable subclass of submodular cost functions called *discounted cost functions*.

The results presented in this section are based on joint work with Gagan Goel, Chinmay Karande and Lei Wang [23, 26].

CHAPTER II

ONLINE BIPARTITE MATCHING WITH UNKNOWN DISTRIBUTIONS

2.1 Introduction

In this chapter we will study the online bipartite matching problem. This is a central problem in algorithms and has been recently found to be an important and useful model for allocation of ad space to advertisers. In its basic form, the problem involves a bipartite graph $G(L, R, E)$, with one side L (ads, jobs, or items to sell, in different motivating examples) known beforehand to the algorithm, and vertices from the other side R (ad-slots, job-candidates, or buyers) arriving one by one online. When a vertex $r \in R$ arrives, its incident edges are revealed, and the algorithm can match it to some currently unmatched neighbor in L. The objective is to maximize the size of the matching obtained at the end.

The online bipartite matching problem can be studied in different input models based on the amount of information available about the vertices in R.

Adversarial: The strictest model is the *adversarial order* model which means that the algorithm knows L but has no information about R (and therefore E) – this is the standard model for online algorithms. In this model, a simple Greedy algorithm achieves a competitive ratio of $1/2$ since it produces a maximal matching, and this is optimal among deterministic algorithms. A simple randomized algorithm which matches the arriving vertex to a random unmatched neighbor does no better. In a beautiful result [37], Karp, Vazirani and Vazirani described an optimal randomized algorithm, which achieves an expected competitive ratio of $1 - 1/e \simeq 0.632$ (see also, [6, 24] for different proofs for the same result). This algorithm, called RANKING,

uses *correlated* randomness to match arriving vertices: pick a random permutation of L, and match each arriving vertex $r \in R$ to the neighbor with the highest rank in the permutation.

Known Distribution: In Feldman et al. [22], the authors introduced a distributional input model, which we shall call the *known distribution input model*. In this model, the algorithm knows beforehand, a base graph $\widehat{G}(L, \widehat{R}, \widehat{E})$, and a distribution $\mathcal{D}$ on $\widehat{R}$. The arriving vertices are sampled i.i.d. (without replacement) from $\widehat{R}$ according to $\mathcal{D}$. Each arriving vertex has the same incident edges as its copy in $\widehat{G}$. Thus this is a weaker model, since the algorithm is guaranteed i.i.d. samples, and also knows the underlying distribution. Clearly, RANKING achieves $1 - 1/e$ in this model as well, since it does so in the adversarial model (for any sequence of inputs, without any prior information). Feldman et al. [22] provided an algorithm which achieves a factor strictly greater than $1 - 1/e$, of $\frac{1-2/e^2}{4/3-2/3e} \simeq 0.670$. This was the first algorithm to beat the $1 - 1/e$ barrier, albeit in a much weaker model.

Unknown Distribution / Random Order: We will study a model that lies in between these two models in terms of how much information about the arriving vertices the algorithm has beforehand. In this model, which we call the *unknown distribution model*, the arriving vertices are guaranteed to be picked from a distribution on some base graph, but the algorithm has no knowledge about the base graph and the distribution. Closely related to this model, but stricter than this model, is the *random order arrival model*, in which there is a base graph $G(L, R, E)$, the algorithm only knows L, and the arriving vertices are guaranteed to be in a random permutation of R. It can be shown that an algorithm which achieves a competitive ratio of α in the random order model also has a factor of α in the unknown distribution model (See Section 2.2.2 for a proof).

The only previous result in this model follows, in fact, from the structure of the RANKING algorithm. The analysis for RANKING in the adversarial model itself

implies that a simple GREEDY algorithm (with consistent tie-breaking) achieves a competitive ratio of $1-1/e$ in the random order model. Clearly, RANKING itself also achieves at least $1-1/e$, since it does so on any input sequence and without any prior information. However, unlike in the case for the known distribution model result (in which [22] broke through the $1-1/e$ 'barrier'), and despite considerable effort, there was no algorithm known to achieve more than $1-1/e$ factor in the unknown distribution model or in the random order model.

2.1.1 Our Results

We analyze the competitive ratio of the RANKING algorithm in the random order arrival model. We prove the following three results:

(a) RANKING achieves a factor of at least 0.653 in the random order model (and hence in the unknown distribution model)[1].

(b) There are graphs for which RANKING achieves a ratio of no more than 0.727 in the random order model.

(c) RANKING achieves a factor of at least $1-\sqrt{\frac{1}{k}-\frac{1}{k^2}+\frac{1}{n}}$ for graphs which have at least $k>1$ disjoint perfect (or near-perfect) matchings.

Remark 1 No algorithm crossing the $1-1/e$ barrier in the unknown distribution or the random order input models was previously known. This answers an open question in [24].

Remark 2 Since the random order model is stricter than the unknown distribution model, the positive results (a and c) hold there as well.

Remark 3 The third result (c) above states that if the graph has certain redundancy with respect to optimal matchings then RANKING is almost optimal! If

[1]We can in fact prove a slightly stronger factor of 0.667 for this problem, but this requires us to use computational means to minimize a non-convex function over fixed number of variables.

the underlying distribution is such that with high probability any instance drawn from the distribution has $\omega(1)$ disjoint matchings of size at least $n - o(n)$ then the RANKING algorithm achieves a factor of $1 - o(1)$. We note that this result is stronger than the result in [4], which provides an algorithm which achieves $1 - O(1/\sqrt{d})$ for d-regular graphs (noting that d regular graphs have d disjoint perfect matchings).

As a additional remark, result (c) resolves a puzzling mystery which is observed during simulations of RANKING on the graph with the upper triangular matrix as its adjacency matrix. If the input order is adversarial then the ratio is $1 - 1/e$ (this is the tight example for the algorithm in that model and can be proved analytically [37]), but with random order input, the ratio is observed to go to 1 as n increases. The mystery is resolved by observing that this graph indeed has many ($\omega(1)$) almost-perfect matches.

Remark 4 We point out that in the unknown distribution model the support for the distribution may be exponentially large, whereas all previous work in the known distribution case assumes that the support is polynomially bounded. So, e.g., our model allows a distribution on a base graph which is of exponential size ($\widehat{R}$ is exponential in size), and an exponential sized distribution on this graph. The algorithm only uses the guarantee that the arriving sequence of n vertices is sampled i.i.d. from some distribution.

2.1.2 Related Models and Results

As discussed earlier, Karp et. al. [37] gave the RANKING algorithm for the online bipartite matching problem in the adversarial model, which is the model with the least amount of information. They showed that RANKING attains a factor of $1 - 1/e$ and established that this is optimal in the adversarial model.

For the known distribution model, which is the model with the most information, Feldman et. al. [22] gave an algorithm that beats the $1 - 1/e$ bound to attain

a factor of $\frac{1-2/e^2}{4/3-2/3e} \simeq 0.670$. They also showed that the optimal factor for any online algorithm in this setting was bounded away from 1. Their upper and lower bounds were subsequently improved by Bahmani and Kapralov [4] to 0.902 and 0.699 respectively in the same setting. They also presented a randomized algorithm that achieves a competitive ratio of $1 - O(1/\sqrt{d})$ for d-regular graphs. Subsequently Mahshadi et. al. [46] provided an improved algorithm with a factor of 0.702 and proved an upper bound of 0.823. Furthermore, it was proved in [24] that no online algorithm can achieve competitive ratio better than 5/6 for any of these models. We summarize the above discussion in Table 1.

Table 1: Summary of Results for Online Bipartite Matching in Various Models

Model	Adversarial Input	Known Distribution	Unknown Distribution
Lower Bounds (algorithms)	$1 - \frac{1}{e}$ [37]	0.67 [22] 0.699 [4] 0.702 [46]	$1 - \frac{1}{e}$ [37] 0.653 [This Chapter] , [44]
Upper Bounds (hardness)	$1 - \frac{1}{e}$ [37]	0.998 [22] 0.902 [4] 0.823 [46]	5/6 [24] 0.823 [46]

Besides the results mentioned in earlier, there has been considerable interest in online matching and allocation problems over the last few years and several variants and generalizations of the bipartite matching problem have been studied. The case of weighted edges was considered by Korula and Pal in [39] and they design an 8-competitive algorithm for this problem. The variant where there are weights on the incoming vertices was studied by Aggarwal et al. in [1]. The authors present tight $1 - 1/e$ factor algorithms for this problem.

Simultaneously and independent of the result described in this chapter, Mahdian and Yan [44] also analyzed the performance of RANKING in the random order model, and also breached the barrier of $1 - 1/e$ by showing that RANKING achieves a factor of at least 0.696. They find a family of strongly factor revealing LPs for this problem and solve large LPs in this family computationally to provide a good lower bound on

the factor.

2.2 Preliminaries

2.2.1 Problem Statement

In this model there is a fixed bipartite graph $\widehat{G}(L, \widehat{R}, \widehat{E})$; one side($L$) of the graph corresponds to a fixed set of vertices and the other side($\widehat{R}$) represents the set of all possible vertices that may arrive online. There is also an (unknown) distribution $\mathcal{D}$ over the vertices of $\widehat{R}$. At each time step, a vertex in $\widehat{R}$ is sampled independently from $\mathcal{D}$ (with replacement), and it needs to be matched to an unmatched neighboring vertex in L upon its arrival. Thus the sample space consists of the different sequences of vertices obtained by drawing n times from the fixed but unknown distribution. We call this the unknown distribution model (UD Model). The goal is to maximize the expected size of the matching.

Let $\mathcal{A}$ be an (possibly randomized) online algorithm. For any realization graph G based on n random samples from $\mathcal{D}$, let $ALG(G)$ be the expected size of the matching produced by $\mathcal{A}$. Let $OPT(G)$ be the size of the largest matching in G. The competitive ratio for $\mathcal{A}$ is defined to be $E_G\left[\frac{ALG(G)}{OPT(G)}\right]$, where the expectation is over the realization graphs.

2.2.2 From Unknown Distribution to Random Order Arrival Model

In this section we define a different, stricter model for online bipartite matching called the Random Order Arrival Model (ROA Model), and establish its relationship with the Unknown Distribution Model (UD Model). As before, in this model, there is a fixed bipartite graph $G(L, R, E)$ with vertex set $L \cup R$. The vertices of L are known in advance, while the vertices of R arrive in random order. Whenever a vertex arrives its incident edges are revealed, and it can be matched off to one of its available (unmatched) neighbors in R. The goal is to maximize the expected size of the matching. The ROA Model can be thought of as sampling from the uniform

distribution on R without replacement. In the next lemma we show that the ROA Model is in fact stricter than the UD Model.

Lemma 1. *Any online algorithm $\mathcal{A}$ that achieves a competitive ratio of α in the ROA Model also achieves a competitive ratio of at least α in the UD Model. Lower bounds for the performance of online algorithms in the UD Model carry over to the ROA Model.*

Proof. Consider a problem instance in the UD Model, with n arriving vertices. Divide the sample space for this instance into classes, each of size $n!$, such that for each class, the multi-set of arriving vertices (from $\widehat{R}$) is the same for every sequence in that class. Each class consists of all the permutations of some set, and furthermore, the probability of occurrence of each sequence in a class is the same. Thus each class can be thought of as an instance of a random order arrival input. Since $\mathcal{A}$ has a competitive ratio of α in the ROA Model, it performs at least as well for all samples in a particular class. Taking expectation over the different classes we get the first part of the lemma. The second part can be proved by a similar argument. □

Hence we will now focus our attention on designing an online algorithm for the Random Order Arrival Model.

2.2.3 Definitions in the ROA Model

Throughout this chapter, for convenience, we will assume that both bi-partitions L and R, have equal size n, and the underlying graph has a perfect matching which we will refer to as OPT (if there are multiple perfect matchings then we choose one arbitrarily). It is an easy exercise to verify that this assumption is without loss of generality. We will use $u, v \in [n]$ to denote the vertices of R(the streaming side) and $s, t \in [n]$, to index the the vertices of R. For any $u \in R$, we will use u^* to denote its match in OPT we will refer to u^* (resp. u) as the *partner* of u (resp. u^*) with respect to OPT.

We also use the notion of time: time t will denote the event when the t^{th} vertex of R is revealed to the algorithm. Define Ω_R (resp. Ω_L) to be the set of all permutations of R (resp. L). For any permutation ρ of the vertices, we will use $\rho(t)$ to denote the vertex at the t^{th} position in ρ and $\rho^{-1}(u)$ to denote the position of the vertex u in ρ. For vertices u, v we say u is above v in ρ if $\rho^{-1}(u)$ is less than $\rho^{-1}(v)$. We can similarly define the notion of a vertex u being below another vertex v. For any permutation $\rho \in \Omega_L$ let $\rho[u \rightsquigarrow s]$ denote the permutation obtained by moving u to position s keeping the order of the other vertices unchanged. We will use $\pi \in \Omega_R$ to represent the order in which the vertices of R arrive.

The RANKING **Algorithm:** We will analyze RANKING which was first proposed in [37]. The algorithm takes an ordering of R as input and produces a matching as its output. The algorithm is described below.

RANKING

1. Choose a random permutation ρ from Ω_L uniformly at random.
2. Apply permutation ρ to L - thereby assigning each vertex a priority or rank.
3. For each arriving vertex from R match it to the vertex (if any) of L, with the highest rank.

By a slight abuse of notation we will use $Ranking(\rho, \pi)$ to denote an invocation of the above algorithm where ρ is the permutation chosen in the first step and π denotes the arrival order of the vertices in R. For any $t \in [n]$ define x_t to be the probability that the vertex $\rho(t) \in L$ at position t get matched in RANKING, where the probability is taken over the random choices of ρ and π.

Events, $B, \widehat{B}, P$-events: For each $\rho \in \Omega_L,\ \pi \in \Omega_R$ and every vertex $v \in L$ (or $v^* \in R$), we define an *event* to be the tuple (ρ, v, π) (or (ρ, π, v^*)). Thus an event is simply the specification of what the two permutations are and which vertex we are

talking about. Note that an event specifies the position of the vertex and its optimal match.

An event (ρ, v, π) is called a B-event if v is matched in $Ranking(\rho, \pi)$, and furthermore it is matched to some vertex $u^* \neq v^*$ such that $\pi^{-1}(u^*) > \pi^{-1}(v^*)$ (i.e., v^* is ranked higher than u^* in π). Informally, a B-event is an event in which v is matched 'below' v^*. Note that this also means that v^* is itself matched.

Symmetrically, an event (ρ, π, v^*) is called a $\hat{B}$-event if v^* is matched in $Ranking(\rho, \pi)$, and furthermore it is matched to some vertex $u \neq v$ such that $\rho^{-1}(u) > \rho^{-1}(v)$ (i.e., v is ranked higher than u in ρ).

An event (ρ, v, π) is called a P-event if v is matched to v^* in $Ranking(\rho, \pi)$. An event (ρ, π, v^*) is called a $\hat{P}$-event if v^* is matched to v in $Ranking(\rho, \pi)$. We define B (resp. $\hat{B}$, resp. P) to be the total probability of B-events (resp. $\hat{B}$-events, resp. P-events).

2.2.4 Symmetry of RANKING in the ROA Model

In this section we will present two important lemma's that distinguish the behavior of Ranking in ROA Model from the adversarial model considered in [37]. The first lemma states that the roles of the streaming and the static bipartitions may be switched without altering the outcome of the algorithm. The second lemma asserts that for the purpose of analysis, without loss of generality we may assume that the given graph is symmetric.

Lemma 2. *For a given graph $G(L, R, E)$ and for a fixed $(\rho, \pi) \in \Omega_L \times \Omega_R$, the output of $Ranking(\rho, \pi)$ on G is same as the output of $Ranking(\pi, \rho)$ on $G'(R, L, E)$ (where G' is obtained from G by switching the two partitions).*

Proof. Let M be the output for $Ranking(\rho, \pi)$ for the graph G. Let us simulate $Ranking(\pi, \rho)$ over G'. We will prove the above claim by induction on the number of vertices in the set L from G' which have arrived at any moment. Suppose the part of

Ranking(π, ρ) constructed over $\rho(1), ..., \rho(t-1)$ in our simulation is consistent with M. Let $v = \rho(t) \in L$ be the t'th vertex to arrive. If possible let v be unmatched in M and suppose it gets matched to $u^* \in R$ in our simulation. By our hypothesis, u^* must have been matched below position t in M. This yields a contradiction since, u^* could have matched a higher vertex in M namely v. Similarly, we can argue the case when v is unmatched in our simulation, but is matched in M.

The only other possibility is that v matches different vertices in M and in our simulation. Suppose v is matched to w^* in M and u^* in our simulation. If $\pi^{-1}(w^*) < \pi^{-1}(u^*)$, then w^* is also matched, to say z, in our simulation. If $\rho^{-1}(z) < t$, then this contradicts the induction hypothesis. On the other hand if $\rho^{-1}(z) > t$, then v should have matched w^* and not u^*. A similar argument works for the case when $\pi^{-1}(w^*) > \pi^{-1}(u^*)$. □

Lemma 3. *Without loss of generality, we may assume that the worst example for* Ranking *in the ROA Model is symmetric.*

Proof. Let $W(L, R, E)$ be a worst example for Ranking in the ROA Model. Let $W_1(L_1, R_1, E_1)$ and $W_2(L_2, R_2, E_2)$ be two copies of W. Consider the graph $W'(L', R', E')$ where $L' = L_1 \cup R_2$, $R' = L_2 \cup R_1$ and $E' = E_1 \cup E_2$. Note that W' is a symmetric graph. Since the two copies are disjoint the competitive ratio for Ranking on W' is just the average competitive ratio for the two components. By Lemma 2, both W_1 and W_2 attain the same competitive ratio. Hence there exists a symmetric graph W' for which Ranking attains its worst competitive ratio. □

Lemma 3 yields the following corollary.

Corollary 4. *Without loss of generality, we may assume that in the worst example for* Ranking *in the ROA Model,* $B = \widehat{B}$.

2.3 Analysis of the RANKING Algorithm

Earlier we mentioned the components missing from previously known analyses of RANKING, *viz.* the B-events and P-events. If the order of arrival is arbitrary, they are irrelevant, because RANKING produces almost no B-events and P-events when faced with the tight example of the complete upper triangular matrix. In this section we prove that these components gain significance in the random arrival model, and help push the approximation factor beyond $\left(1 - \frac{1}{e}\right)$.

In Section 2.3.1, we consider the special case when the aggregate probability of a P-event is small and prove a bound on the competitive ratio of RANKING as a function of this probability. This implies the near optimal result in the case with many disjoint matchings. In Section 2.3.2, we prove a factor of 0.653 in general.

First we prove a simple but important counting lemma which expresses the gains of the algorithm above 1/2 in terms of the B, $\hat{B}$ and P-events. In fact the lemma holds for RANKING based on any distribution of ρ and π (even for fixed ρ, π).

Lemma 5.

$$ALG \geq \frac{n}{2} + \frac{B}{2} + \frac{\hat{B}}{2} + \frac{P}{2}$$

Proof. We count the total number of vertices matched and divide by two to get the size of the matching. We have fixed an OPT; for every pair v, v^* in the OPT matching, we know that at least one of them is matched in ALG (in every ρ, π). The greedy property of RANKING implies that (ρ, π, v) and (ρ, π, v^*) cannot simultaneously be B-event and $\hat{B}$-event respectively. If both v and v^* are matched, we pick the vertex which is not a B-match (resp. $\hat{B}$-match). If v and v^* are matched to each other in a P-event, then we pick v (to break ties). This gives us n matched vertices. Now, each B-match and $\hat{B}$-match has not been already counted in these n vertices. So also, each P-event corresponds to two matched vertices, but we counted only the left vertex. Thus we can add $B + \hat{B} + P$ to n to get a lower bound on the total number

of matched vertices. This proves the lemma. □

From the statement of Lemma 5, it is clear that we need to prove that in random permutation model (with symmetry implied by Corollary 4), B-events, $\hat{B}$-events and P-events make a sizeable combined contribution. We achieve this in the following lemma:

Lemma 6.

$$\sum_t \frac{(t-1)}{n} x_t \;\leq\; \hat{B} + P - \frac{P^2}{2n}$$

where x_t is the probability that the vertex at rank t in L is matched in RANKING.

This lemma is the technical core of our main result. Before moving on to the proof, we first introduce some notation which will be useful.

Notice that the events that contribute to the x_t variables on the LHS are simply the set of all occurrences of matched vertices, whereas the RHS consists of events where $v^* \in R$ is matched to $u \in L$ such that $\rho^{-1}(u) \geq \rho^{-1}(v)$. Our proof involves designing a many-to-many map between the former set of events to the latter.

In order to do that, we need to classify the relevant events into two types with distinct properties. We need the following definitions: If ρ is any permutation on L, then ρ_{-v} is the permutation on $L - \{v\}$ consistent with ρ. As defined in Section 2.2.3, $\rho[v \rightsquigarrow s]$ is the permutation obtained by inserting v into ρ_{-v} at position s. A *column* is a collection of n events defined by a permutation ρ_{-v} (on L) and π (on R), by inserting v into ρ_{-v} at all n positions.

This notion of *columns* of events has been used in related literature without being defined explicitly. For example, it forms the basis of analysis of RANKING in the arbitrary arrival model, as found in [37, 24, 6].

We will use the above property, but in our analysis, we will also look at events in columns where v is always matched, but v^* may be unmatched. From this point of view, we need to classify the columns into two different types.

Definition 1. *A column (ρ_{-v}, π) is said to be a T1 (type 1) column if in the configuration $(\rho[v \rightsquigarrow n], \pi)$,*

- *v is unmatched OR*
- *$(\rho[v \rightsquigarrow n], v, \pi)$ is a B-event OR*
- *$(\rho[v \rightsquigarrow n], v, \pi)$ is an A-event and $(\rho[v \rightsquigarrow n], \pi, v^*)$ is also an A-event*

Otherwise, the column is said to be a T2 (type 2) column, i.e. if in the configuration $(\rho[v \rightsquigarrow n], \pi)$,

- *v is matched to v^* (P-event) OR*
- *$(\rho[v \rightsquigarrow n], v, \pi)$ is an A-event and v^* is unmatched*

We will use the following properties, which are simple consequences of the above definitions and the RANKING algorithm:

- v^* is always matched in a configuration where v is in a T1 column.
- If v is in a T2 column, then v^* can match no higher than v.

We can now prove Lemma 6.

Proof. **(of Lemma 6)** Let $\mathcal{X}$ be the set of all events (ρ, u, π), where $u = \rho(t)$ is matched to $v^* \in R$. Similarly, $\hat{\mathcal{B}}$, $\mathcal{P}$ and $\hat{\mathcal{P}}$ are the sets of all $\hat{B}$-events, P-events and $\hat{P}$-events respectively. Partition $\mathcal{X}$ into two sets: $\mathcal{X}_1$ such that (ρ_{-v}, π) is a T1 column for v and $\mathcal{X}_2$ such that (ρ_{-v}, π) is a T2 column for v.

We now define our many-to-many maps. For $(\rho, u, \pi) \in \mathcal{X}_1$, consider the situation of v^*, in the configuration $(\rho[v \rightsquigarrow s], \pi)$ for $s < t$. In other words, we move v to all positions above u. We claim that $(\rho[v \rightsquigarrow s], \pi, v^*)$ is a either a $\hat{B}$-event or a $\hat{P}$-event. The fact that v^* remains matched in $(\rho[v \rightsquigarrow s], \pi)$ follows from the fact the configuration is a T1 column for v, and Lemma 9. Let w be the vertex to which v^* is

matched in $(\rho[v \rightsquigarrow s], \pi)$. To prove $\rho[v \rightsquigarrow s](w) \geq s$, observe that if we remove v from ρ, v^* may be unmatched, or matched no earlier than its original match u which was at position t. Now, adding v back to ρ_{-v} at any position s above that of u, can improve the position of the match of v^* to no higher than s. This implies that $(\rho[v \rightsquigarrow s], \pi, v^*)$ is a either a $\hat{B}$-event (if $\rho[v \rightsquigarrow s](w) > s$) or a $\hat{P}$-event (if $\rho[v \rightsquigarrow s](w) = s$).

Hence, we can now map each event $(\rho, u, \pi) \in \mathcal{X}_1$ to $t-1$ different events $(\rho[v \rightsquigarrow s], \pi, v^*)$, all which are either $\hat{B}$-events or $\hat{P}$-events. For each $(\rho, u, \pi) \in \mathcal{X}_1$, we define:

$$f(\rho, u, \pi) \;=\; \{\; (\rho[v \rightsquigarrow s], \pi, v^*) \mid 1 \leq s < t \;\} \;\subseteq\; \hat{\mathcal{B}} \cup \hat{\mathcal{P}}$$

Let

$$M \;=\; \biguplus_{(\rho,u,\pi)\in\mathcal{X}_1} f(\rho, u, \pi)$$

where the $\biguplus$ symbol represents multiset union. Clearly, although each $f(\rho, u, \pi)$ is a simple set, the same $\hat{B}$-event (or $\hat{P}$-event) may be part of two different such sets. We will now quantify this over-counting. First, let us partition M into $M_b = M \cap \hat{\mathcal{B}}$ and $M_p = M \cap \hat{\mathcal{P}}$.

Any event $(\rho, \pi, v^*) \in M_b$ appears in the maps of at most n distinct events from $\mathcal{X}_1$, those obtained by moving v to all positions from 1 to n, keeping the rest of the configuration constant. Therefore,

$$|M_b| \leq n|\hat{\mathcal{B}}_1| \tag{1}$$

where $\hat{\mathcal{B}}_1$ is the simple set obtained by discarding duplicates from M_b.

For each $\hat{P}$-event (ρ, π, v^*) in M_p, we can form a set N_p of the corresponding P-events: (ρ, v, π). Let $\mathcal{P}_1$ be the simple set obtained by discarding duplicates from N_p. Our goal is to count the number of times each event $(\rho, v, \pi) \in \mathcal{P}_1$ appears in the multiset N_p. We claim that this number is at most $n - U(\rho, v, \pi)$ where U is defined for a P-event (ρ, v, π) as $U(\rho, v, \pi)$ is the the number of P-events$(\rho[v \rightsquigarrow s], v, \pi)$ in column(ρ_{-v}, π) such that $s < \rho^{-1}(v)$. This follows from the fact that if (ρ, v, π) and

$(\rho[v \rightsquigarrow s], v, \pi)$ are both P-events and $\rho^{-1}(v) < s$ then $(\rho[v \rightsquigarrow s], v, \pi)$ does not appear in $f(\rho, v, \pi)$. Now for notational convenience, we define:

$$P(\rho_{-v}, \pi) = \text{Total number of } P\text{-events in column } (\rho_{-v}, \pi)$$

Therefore, we can now bound the size of M_p as:

$$\begin{aligned}
|M_p| &= |N_p| \\
&\leq \sum_{(\rho,v,\pi)\in\mathcal{P}_1} n - U(\rho, v, \pi) \\
&= n|\mathcal{P}_1| - \sum_{\text{T1}(\rho_{-v},\pi)} \left(\sum_{(\rho,v,\pi)\in(\rho_{-v},p)} U(\rho, v, \pi) \right) \\
&= \left(n \sum_{\text{T1 } (\rho_{-v},\pi)} P(\rho_{-v}, \pi) \right) - \left(\sum_{\text{T1 } (\rho_{-v},\pi)} \frac{[P(\rho_{-v}, \pi)]^2}{2} \right) \qquad (2)
\end{aligned}$$

First let y_t be the probability that an event (ρ, v, π) such that $\rho^{-1}(v) = t$ is in $\mathcal{X}_1$. Using the fact that such an event has $|f(\rho, v, \pi)| = t - 1$, and using equations (1) and (2),

$$\begin{aligned}
\sum_t (t-1) y_t &= \frac{|\hat{M}_1|}{|\Omega_L \times \Omega_R|} = \frac{|M_b| + |M_p|}{|\Omega_L \times \Omega_R|} \\
&\leq \frac{n|\hat{\mathcal{B}}_1|}{|\Omega_L \times \Omega_R|} + \frac{\sum_{\text{T1 } (\rho_{-v},\pi)} \left(nP(\rho_{-v}, \pi) - \frac{[P(\rho_{-v}, \pi)]^2}{2} \right)}{|\Omega_L \times \Omega_R|} \qquad (3)
\end{aligned}$$

Now, we will deal with events in $\mathcal{X}_2$. Since these are events in T2 columns, (ρ, π, v^*) must already be a $\hat{B}$-event or a $\hat{P}$-event. Let $\hat{\mathcal{B}}_2$ be the set of $\hat{B}$-events (ρ, π, v^*) such that v^* is matched to u in the corresponding $(\rho, u, \pi) \in \mathcal{X}_2$. By this one-to-one correspondence, $|\hat{\mathcal{B}}_2| = |\mathcal{X}_2 - \mathcal{P}|$. Let $\mathcal{P}_2 = \mathcal{X}_2 \cap \mathcal{P}$. Therefore, $|\mathcal{X}_2| = |\hat{\mathcal{B}}_2| + |\mathcal{P}_2|$.

Since the probability that $(\rho, u, \pi) \in \mathcal{X}_2$ is $x_t - y_t$, we have:

$$\sum_t (x_t - y_t) = \frac{|\hat{\mathcal{B}}_2|}{|\Omega_L \times \Omega_R|} + \frac{|\mathcal{P}_2|}{|\Omega_L \times \Omega_R|} \qquad (4)$$

Let $\mathcal{Q}$ be the multiset formed by including each event $(\rho, v, \pi) \in \mathcal{P}_2$ a total of $\rho^{-1}(v)$ times. Then we can multiply the t'th term on the LHS of equation (4) by

$(t-1)$ and compensate for the over-counting by (a) raising the coefficient of $|\hat{\mathcal{B}}_2|$ to n and (b) by using $|\mathcal{Q}|$ instead of $|\mathcal{P}_2|$

$$\sum_t (t-1)(x_t - y_t) \;\leq\; \frac{n|\hat{\mathcal{B}}_2|}{|\Omega_L \times \Omega_R|} + \frac{|Q|}{|\Omega_L \times \Omega_R|} \tag{5}$$

Now we will bound $|\mathcal{Q}|$. Every P-event $(\rho, v, \pi) \in \mathcal{P}_2$ where appears in $\mathcal{Q}$ a total of $\rho^{-1}(v)$ times. Now $\rho^{-1}(v) = n - (n - \rho^{-1}(v))$. But for every P-event in a T2 column, there are $n - \rho^{-1}(v)$ P-events $(\rho[v \rightsquigarrow s], v, \pi)$ at positions $s > \rho^{-1}(v)$ in the same column. Therefore, we can say that each event in $(\rho, v, \pi) \in \mathcal{P}_2$ appears $n - W(\rho, v, \pi)$ times in $\mathcal{Q}$, where W is defined for $(\rho, v, \pi) \in \mathcal{P}_2$ as $W(\rho, v, \pi)$ is the number of P-events $(\rho[v \rightsquigarrow s], v, \pi)$ in column (ρ_{-v}, π) such that $s > \rho^{-1}(v)$.

And finally, we will borrow the notation $P(\rho_{-v}, \pi)$ defined earlier to mean the total number of P-events in the column.

Following arguments analogous to the proof of equation (3), we can now rewrite equation (5) as:

$$\sum_t (t-1)(x_t - y_t) = \frac{n|\hat{\mathcal{B}}_2|}{|\Omega_L \times \Omega_R|} + \frac{\sum_{\text{T2 } (\rho_{-v}, \pi)} \left(nP(\rho_{-v}, \pi) - \frac{[P(\rho_{-v}, \pi)]^2}{2} \right)}{|\Omega_L \times \Omega_R|} \tag{6}$$

Next, we observe that $\hat{\mathcal{B}}_1$ and $\hat{\mathcal{B}}_2$ are necessarily disjoint, since the events in $\hat{\mathcal{B}}_1$ have v in a T1 column and events in $\hat{\mathcal{B}}_2$ have v in a T2 column. Adding equations (3) and (6),

$$\sum_t (t-1)x_t \leq \frac{n(|\hat{\mathcal{B}}_1| + |\hat{\mathcal{B}}_2|)}{|\Omega_L \times \Omega_R|} + \frac{\sum_{\text{Column } (\rho_{-v}, \pi)} \left(nP(\rho_{-v}, \pi) - \frac{[P(\rho_{-v}, \pi)]^2}{2} \right)}{|\Omega_L \times \Omega_R|}$$

Now, the sum of $P(\rho_{-v}, \pi) - \frac{[P(\rho_{-v}, \pi)]^2}{2}$ over all columns is maximized when $P(\rho_{-v}, \pi)$ is equal over all columns. This follows from the fact that the sum of squares of k numbers with fixed sum is minimized when they are all equal. Therefore, equalizing the value of $P(\rho_{-v}, \pi)$ over all columns, we arrive at:

$$\sum_t (t-1)x_t \;\leq\; \frac{n|\hat{\mathcal{B}}|}{|\Omega_L \times \Omega_R|} + \frac{n|\mathcal{P}|}{|\Omega_L \times \Omega_R|} - \frac{|\mathcal{P}|^2}{2|\Omega_L \times \Omega_R|^2}$$

$$= \quad n\hat{B} + nP - \frac{P^2}{2}$$

□

2.3.1 The Case with Few Perfect Matches

In this section, we will consider the case that RANKING produces very few perfect matches. In fact, there exists a class of graphs for which this property holds: Graphs with many disjoint perfect matchings. If the graph has k disjoint perfect matchings, then there exists a perfect matching so that RANKING produces at most $\frac{1}{k}$ perfect matches.

Theorem 7. *Defining p as the aggregate probability of getting a perfect match (according to some fixed optimal matching),* RANKING *achieves a competitive ratio of $1 - \sqrt{p - p^2 + \frac{1}{n}}$ in the random order input model.*

Proof. From Lemma 5, and Corollary 4 we have

$$ALG \ = \sum_t x_t \geq \ \frac{n}{2} + \widehat{B} + \frac{P}{2} \tag{7}$$

Define $a = \frac{ALG}{n}$, which is the final competitive ratio and $p = \frac{P}{n}$, the aggregate probability of a perfect match. From Lemma 6 we have:

$$\sum_t \frac{t-1}{n} x_t \leq \widehat{B} + P - \frac{P^2}{2n}$$

Now we minimize $\sum_t \frac{t-1}{n} x_t$ by simply top aligning all the $\sum_t x_t = ALG$, to give

$$\sum_t \frac{t-1}{n} x_t \ \geq \ \frac{ALG(ALG-1)}{2n}$$

So we have $\widehat{B} \ \geq \ \frac{ALG(ALG-1)}{2n} - P + \frac{P^2}{2n}$. Substituting in equation (7) we get:

$$ALG \ \geq \ \frac{n}{2} + \frac{ALG(ALG-1)}{2n} - \frac{P}{2} + \frac{P^2}{2n}$$

$$\Rightarrow 2a \ \geq \ 1 + a^2 - a/n - p + p^2$$

$$\Rightarrow (1-a)^2 \;\leq\; p - p^2 + a/n$$
$$\Rightarrow a \;\geq\; 1 - \sqrt{p - p^2 + \frac{1}{n}}$$

□

Corollary 8. *If a graph has $k > 1$ disjoint perfect matchings, then* RANKING *achieves a competitive ratio of at least* $1 - \sqrt{\frac{1}{k} - \frac{1}{k^2} + \frac{1}{n}}$

Proof. There exists a perfect matching $\mathcal{M}$ such that RANKING makes at most n/k perfect matches according to $\mathcal{M}$. □

2.3.2 The General Case

If the aggregate probability of perfect matches p equals $1/2$, then the previous result does not say much (a ratio of $1/2$). In this section we prove a more interesting bound on $\sum_t \frac{t-1}{n} x_t$ which will help us prove a bound which beats $1 - 1/e$ in general.

The following lemma gives us a sense of the distribution of x_t and would be helpful in proving the main result in this section

Lemma 9.

$$\forall\ t: \quad 1 - x_t \;\leq\; \frac{\sum_{s \leq t} x_s}{n}$$

Proof. (Sketch) For fixed ρ and π consider the execution $Ranking(\rho, \pi)$. Suppose $v = \pi(t)$ does not get matched in this simulation. Then clearly v^* must have been matched, otherwise v and v^* would have matched each other. In particular, observe that v^* should have matched to a vertex, say u, arriving before v.

Let $\Pi = \{\pi\,[v \rightsquigarrow s] \mid \forall s \in [n]\}$. We claim that, for all $\tilde{\pi} \in \Pi$, v would be matched no lower than position t in $Ranking(\rho, \tilde{\pi})$. This is because if v is moved to a position below $\pi^{-1}(u)$ then v^* is already matched by the time v arrives. On the other hand, moving v to a higher position than $\pi(u)^{-1}$ only increases the options available to v^*, thus can only increase its likelihood of getting matched.

Let $\mathcal{I}_\pi^\rho(t)$ be the indicator variable for the event that $\pi(t)$ is missed in $Ranking(\rho, \pi)$. Thus from the above arguments we can define a map that takes a *miss event* at t to n matches belonging to the set of matches above position t i.e.

$$n\left[\sum_{\rho,\pi} 1 - \mathcal{I}_\pi^\rho(t)\right] \leq \sum_{s\leq t}\sum_{\rho,\pi} \mathcal{I}_\pi^\rho(s)$$

Normalizing the above equation proves the lemma. □

We now move on to the main result of this section - a bound on $\sum_t \frac{t-1}{n} x_t$.

Lemma 10.

$$\sum_t \frac{(t-1)x_t}{n} \geq 0.26n - o(1)$$

Proof. From Lemma 9, we have: $\forall\ t:\ \ 1 - x_t \ \leq\ \frac{\sum_{s\leq t} x_s}{n}$. Defining $LB_t := \frac{1-\frac{\sum_{s<t} x_s}{n}}{(1+\frac{1}{n})}$ and rearranging, we have

$$\forall\ t:\ x_t \geq LB_t$$

Thus, we look for the optimal solution to the following linear program:

$$\left\{ min \sum_t \frac{t x_t}{n} \ \text{ s.t. } \forall\ t,\ \ x_t \geq LB_t \right\}$$

In the program, we have ignored the lower order term $\sum_t x_t/n$. We now define an operation which takes one feasible solution to another: If there is a t such that $x_t < 1$ and $x_t > LB_t$, and if there is an $s < t$ s.t. $x_s < 1$, then we reduce x_t and increase x_s by the same amount, equal to $\min\{x_t - LB_t, 1 - x_s\}$. This operation keeps the solution feasible:

- for $r < s$, there is no change in either x_r or in LB_r.
- for $s \leq r < t$, x_r goes up and LB_r stays the same.
- x_t drops to a value at least the original LB_t and LB_t goes down.
- for $r > t$, there is no change in either x_r or LB_r.

Furthermore, it is clear that the objective function value goes down.

Given a feasible solution $\{x_t\}$, we repeatedly apply this operation to obtain another feasible solution with lower value, until we can not apply it any more. This proves that the optimal solution has the following form: For some $t^* \in [1, n]$,

- for all $s < t$, $x_s = 1$
- $LB_t \leq x_t \leq 1$
- for all $s > t$, $x_t = LB_t$.

Thus we know the form of the solution, now we can minimize over t^*. Minimizing the resulting function gives $\sum_t \frac{tx_t}{n} \geq 0.26n$. □

Theorem 11. RANKING *achieves a competitive ratio of at least 0.653 in the random order input model.*

Proof. This proof is a generalization of the proof of Theorem 7. From Lemma 5 and Corollary 4 we have

$$ALG \;\geq\; \frac{n}{2} + \widehat{B} + \frac{P}{2} \tag{8}$$

Define $a = \frac{ALG}{n}$, which is the final competitive ratio, and define $p = \frac{P}{n}$, the probability of a perfect match. From Lemma 6 and Lemma 10 we obtain:

$$0.26n \;\leq\; \sum_t \frac{t-1}{n} x_t \;\leq\; \widehat{B} + P - \frac{P^2}{2n}$$

Substituting in equation (8) we get:

$$\begin{aligned} ALG \;&\geq\; \frac{n}{2} + 0.26n - P + \frac{P^2}{2n} + \frac{P}{2} \\ \Rightarrow a \;&\geq\; \frac{1}{2} + 0.26 - \frac{p}{2} + \frac{p^2}{2} \end{aligned} \tag{9}$$

Also, from 8 we have

$$a \geq \frac{1}{2} + \frac{p}{2} \tag{10}$$

Minimizing the maximum of the two bounds obtained in (9) and (10) we get $a \geq 0.653$ (for $p \simeq 0.3$). □

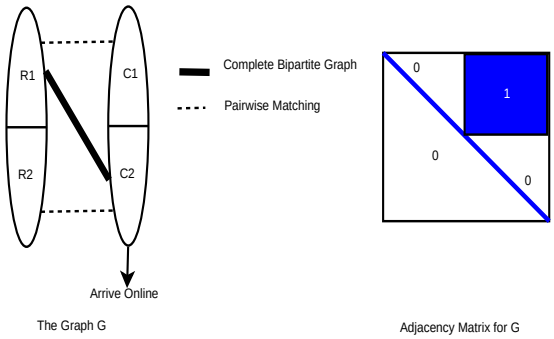

Figure 1: The 0.75 example

2.4 Upper Bounds

In this section, we introduce two families of graphs which provide upper bounds of 0.75 and 0.727 on the competitive ratio of RANKING in the ROA model. The former has a simpler analysis, and hence, we present it first.

2.4.1 An Example where RANKING Achieves 0.75

In this section we will present and analyze a graph for which the RANKING algorithm achieves a factor of 0.75. This will serve as a warm up for the more involved analysis of the example for which the algorithm attains a factor of 0.727.

Let G be the bipartite graph over n vertices whose adjacency matrix A is defined below. The graph is shown in figure 1.

$$A[i][j] \;=\; \begin{cases} 1 & \text{if } i = j \\ 1 & \text{if } i < n/2 \;,\; j > n/2 \\ 0 & \text{otherwise} \end{cases}$$

The rows and columns of A represent the two partitions of vertices of G. For the purpose of the algorithm we will assume that vertices corresponding the columns of A arrive in random order, while those represented by the rows are shuffled according to a uniform random permutation.

For the analysis we will partition the vertices of G in to the following four sets.

Let C_1 be the vertices corresponding to columns 0 through $n/2$ and let C_2 be the vertices for rest of the columns. Similarly let R_1 be the vertices represented by rows 0 through $n/2$ and R_2 be the remaining vertices. For any $v_t \in C_1 \cup C_2$ we will use v_t^* to denote the partner of v_t in the optimal perfect matching. At any time, t let $H(t)$ be the number of unmatched vertices in C_1. This includes vertices in C_1 which are yet to arrive by time t and those vertices of C_1 that could not be matched upon arrival. Define $F(t)$ to be the number of unmatched vertices in R_1 at time t.

Note that all vertices in C_2 will surely get matched, but some vertices in C_1 may not get matched if their unique neighbor in R_1 gets matched to a vertex in C_2. Thus the factor for the algorithm is $\left(|C_1| + |C_2| - E\left[H(n)\right]\right)/n = 1 - E\left[H(n)/n\right]$.

In Lemma 12 we represent the expected behavior of $H(t)$ in terms of $E[F(t)]$. In Lemma 13 we find an explicit formula for the expected behavior of $F(t)$. Combining these two lemmas gives us a differential equation representing the limiting behavior of $E[H]$.

For the rest of the analysis we will only consider the vertices that arrive in the interval $[0, n - n^{0.9}]$. This is done to make the analysis amenable to concentration bounds. Furthermore, ignoring the last $n^{0.9}$ vertices only affects the competitive ratio by a $o(1)$ factor.

Lemma 12. *For all $t \in [0, n - n^{0.9}]$,*

$$E\left[H(t+1) - H(t)\right] \geq -\frac{E\left[F(t)\right]}{n-t} - o(1/n)$$

Proof. Let v_{t+1} be the vertex arriving at time $t+1$. H reduces by 1 at time $t+1$ if v_{t+1} belongs to C_1 and its unique neighbor in R_1 is still unmatched. Let $Q(t)$ be the number of vertices in C_1 that are yet to arrive by time t. By Chernoff bounds $Q(t)$ is tightly concentrated around its expected value of $\frac{n-t}{2}$ i.e. $Q(t)$ is less than $\frac{(n-t)}{2} - 2\sqrt{n \log n}$ with probability at most $1/n^2$. Of the $Q(t)$ vertices in C_1 that are yet to arrive, $F(t)$ are such that their unique neighbor in R_1 is still unmatched.

Hence,

$$\begin{aligned}
E\left[H(t+1)-H(t)\right] &= -\Pr[v_{t+1}\in C_1,\ v^*_{t+1}) \text{ is still unmatched}] \\
&= -\frac{1}{2}\Pr[\ v^*_{t+1}) \text{ is still unmatched}|v_{t+1}\in C_1] \\
&= -\frac{1}{2}\frac{E[F(t)]}{Q(t)} \\
&\geq -\frac{1}{2}\frac{E[F(t)]}{(n-t)/2-2\sqrt{n\log n}}+\Pr\left[Q(t)<\frac{n-t}{2}-2\sqrt{n\log n}\right] \\
&\geq -\frac{E[F(t)]}{n-t}-o(1/n)
\end{aligned}$$

□

Lemma 13. *For all $t\in[0,n-n^{0.9}]$,*

$$E\left[F(t+1)-F(t)\right]\leq -2\frac{E\left[F(t)\right]}{n-t}+o(1/n)$$

Proof. Let v_{t+1} be the vertex arriving at time $t+1$. If v_{t+1} belongs to C_1 then F reduces by 1 iff H falls by 1 at time $t+1$. By the proof of Lemma 12 this happens with probability at most $\frac{E[F(t)]}{n-t}+o(1/n)$. If v_{t+1} belongs to belongs to C_2 then F reduces by 1 only if v_{t+1} is matched to a vertex in R_1. Next we analyze the probability of this event.

Consider $u^*\in R_2$ and let u be its unique neighbor in C_2. Recall that the RANKING algorithm randomly permutes the vertices in $R_1\cup R_2$ and each arriving vertex chooses the highest unmatched vertex in its neighborhood. Thus if u is to match a vertex in R_1, u^* must lie below the highest unmatched vertex in R_1. Consider the pair (u,u^*) such that u is yet to arrive by time t and there is an unmatched vertex in R_1 that is ranked higher than u^*. To bound the probability of this event we note that the vertices of R_1 are matched starting from the top by vertices in C_2, while arriving vertices in C_1 match their only neighbor in R_1 whenever possible. Since we started with a random permutation of vertices of $R_1\cup R_2$ with R_1 constituting half the vertices we can pessimistically estimate the position of the highest unmatched vertex in R_1 by $2*F(t)$.

Therefore the above event happens with probability at least $2E\left[F(t)\right]/n$. Furthermore the probability that u is yet to arrive by time $t+1$ is $1-t/n$. Combining the above two statements, we get $\Pr\left[v_{t+1} \text{ does not match } v^*_{t+1} \mid v_{t+1} \in C_2\right] \geq 2F(t)/(n-t)$. Thus we have,

$$\begin{aligned}
E\left[F(t+1)-F(t)\right] &= \Pr\left[v_{t+1} \text{ matches } v^*_{t+1} \;\&\; v_{t+1} \in C_1\right] \\
&\quad - \Pr\left[v_{t+1} \text{ doesn't match } v^*_{t+1} \;\&\; v_{t+1} \in C_2\right] \\
&\leq -\frac{E\left[F(t)\right]}{n-t} \\
&\quad - \Pr\left[v_{t+1} \text{ doesn't match } v^*_{t+1} \mid v_{t+1} \in C_2\right] \times \Pr\left[v_{t+1} \in C_2\right] \\
&\leq -\frac{F(t)}{n-t} - \frac{1}{2}\frac{2E[F(t)]}{n-t} + o(1/n) \\
&= -\frac{2F(t)}{n-t} + o(1/n)
\end{aligned}$$

□

The statement of Lemma 13 becomes clearer if we scale the time by a factor of n, i.e we let t be the the time when exactly nt vertices have arrived and let $F(t)$ and $H(t)$ be the fraction of unmatched vertices in R_1 and C_1 respectively.

Lemma 13 can now be stated as,

$$\frac{E\left[F(t+1/n)-F(t)\right]}{1/n} = -\frac{2F(t)}{1-t/n} + o(1/n) \tag{11}$$

We claim the random process described by the above equation is well approximated by the trajectory of the differential equation

$$\frac{df}{dt} \leq -\frac{2f}{1-t} + o(1/n) \tag{12}$$

where this equation has been obtained from equation (11) by replacing the right-hand side with the appropriate limiting value as n tends to infinity, dy/dt. This follows easily from known techniques, such as, for example, Kurtz's theorem (Refer [40]). To clarify the connection we state a version of Kurtz's theorem.

Theorem 14 (Kurtz's Theorem [40]). *Suppose we are given a finite set of vectors* $\{e_1, e_2 \cdots e_k\}$ *in* R^d. *We consider an initial process* $\vec{x}\ (t)$ *with generator*

$$Lf(\vec{x}) = \sum_{i=1}^{k} \lambda_i(\vec{x})(f(\vec{x} + \vec{e}_i) - f(\vec{x}))$$

and a scaled process $\vec{z}_n\ (t)$ *with generator*

$$L_n f(\vec{x}) = \sum_{i=1}^{k} n\lambda_i(\vec{x})(f(\vec{x} + \frac{\vec{e}_i}{n}) - f(\vec{x}))$$

The limiting operator L_∞ *satisfies*

$$L_\infty f(\vec{x}) = \sum_{i=1}^{k} n\lambda_i(\vec{x}) \left\langle \nabla f(\vec{x}), \vec{e}_i \right\rangle$$

and corresponds to the deterministic solution $\vec{z}_\infty$ *of the equation*

$$\frac{d}{dt}\ \vec{z}_\infty\ (t) = \sum_{i=1}^{k} \lambda_i(\vec{z}_\infty\ (t))\ \vec{e}_i \tag{13}$$

Let $\lambda_i(\vec{x}) : \mathrm{R}^d \to \mathrm{R}^+$ *be uniformly bounded and Lipschitz continuous, and let* $\vec{z}_\infty$ *be the unique solution of equation* (13) *with* $\vec{z}_\infty\ (0) = \vec{x}\ (0)$. *For each finite* T *there exist a positive constant* α_1 *and a function* α_2 *with*

$$\lim_{\epsilon \downarrow 0} \frac{\alpha_2(\epsilon)}{\epsilon^2} \in (0, \infty) \text{ and } \lim_{\epsilon \uparrow \infty} \frac{\alpha_2(\epsilon)}{\epsilon} = \infty$$

such that, for all $n \geq 1$ *and* $\epsilon > 0$,

$$\Pr\left[\sup_{0 \leq t \leq T} |\ \vec{z}_n\ (t) - \vec{z}_\infty\ (t)| \geq \epsilon \right] \leq \alpha_1 e^{-n\alpha_2(\epsilon)}$$

Moreover, k_1 *and* k_2 *can be chosen independently of* $\vec{x}$.

By applying Kurtz's theorem, we see that as n goes to infinity, the limiting process for equation (11) is given by the differential equation (12).

Solving equation (12) we get $F(t) \leq \frac{(n-t)^2}{2n} - o(1)$. By a similar argument and substituting in equation (14) the limiting process representing the evolution of H is

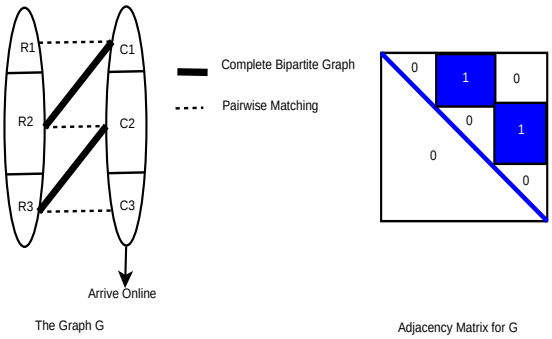

Figure 2: The 0.727 example

given by the following differential equation,

$$\frac{dh}{dt} \geq -\frac{n-t}{2n} + o(1/n) \tag{14}$$

Solving equation (14), we get $H(n) \geq n/4 - o(1)$. Thus the RANKING algorithm achieves a competitive ratio of at most $0.75 + o(1)$ for the above graph.

2.4.2 An Example where RANKING Achieves 0.727

In this section we will sketch the analysis of graph for which the RANKING algorithm achieves a factor of 0.727. Let G be the bipartite graph over n vertices whose adjacency matrix A is defined below. The graph G is shown in figure 2.

$$A[i][j] \;=\; \begin{cases} 1 & \text{if } i = j \\ 1 & \text{if } i < 0.3n \;,\; 0.3n < j \leq 0.7n \\ 1 & \text{if } 0.3n < i \leq 0.7n \;,\; j > 0.7n \\ 0 & \text{otherwise} \end{cases}$$

As before, for the purpose of the algorithm we will assume that vertices corresponding the columns of A arrive in random order, while those represented by the rows are shuffled according to a uniform random permutation.

For the analysis we will partition the vertices of G in to the following six sets. Let C_1 be the vertices corresponding to columns 0 through $0.3n$; let C_2 be the vertices

for the columns $0.3n$ to $0.7n$ and let C_3 be the vertices corresponding to rest of the columns. Similarly let R_1 be the vertices represented by rows 0 through $0.3n$; let R_2 be the vertices corresponding to rows $0.3n$ through $0.7n$ and let R_3 be the remaining vertices.

At any time, t let $F(t)$ be the number of unmatched vertices in C_1. Define $H(t)$ to be the number of unmatched vertices in R_1; $J(t)$ be the number of unmatched vertices in R_2 at time t and let $K(t)$ be the number of unmatched vertices in R_2 whose unique neighbor in C_2 is yet to arrive by time t. By a slight abuse of notation we will use these variables to denote their corresponding sets as well. The exact meaning would be apparent from the context. Also let $SizeH(t) = n - \text{argmin}\,\{\rho^{-1}(v^*) \mid v^* \in H(t)\}$ and define $Last(t) = \text{argmax}\,\{\rho^{-1}(v^*) \mid v^* \in R_2,\ v^* \text{ was matched to a vertex in } C_3\}$.

Note that all vertices in C_3 will surely get matched, but some vertices in C_1 and C_2 may not get matched. By symmetry the number of unmatched vertices in C_2 is same as the number of unmatched vertices in R_2. Thus the factor for the algorithm is $1 - E\left[\frac{F(n)+J(n)}{n}\right]$.

In the subsequent lemmas we will derive equations describing the expected behavior of F and H in terms of the variables defined above. Throughout the rest of the analysis we will assume that at all times $t \in [0, n - n^{0.9}]$, the number of vertices from any of the sets C_1, C_2, C_3 that have arrived is equal to the expected value. This is by a simple application of Chernoff bounds and as we saw in the earlier proof for the upper bound this assumption only adds $o(1/n)$ looseness to any of the difference equations.

We will use lower case letters for each of the discrete variables defined above in the the differential equations corresponding to their difference equations for the processes. This is to highlight the fact that we have approximated discrete stochastic variables by continuous functions in the limit.

Lemma 15. *For all* $t \in [0, n - n^{0.9}]$,

$$E\left[F(t+1) - F(t)\right] \approx -\frac{H(t)}{n-t}$$

In the limit the above random process is closely approximated by the trajectory of the following differential equation.

$$\frac{df}{dt} \approx -\frac{h}{n-t}$$

Proof. The number of vertices belonging to C_1 that are yet to arrive is tightly concentrated around $0.3(n-t)$. Of these, only $H(t)$ are such that their unique neighbor R_1 is still unmatched. The vertex arriving at $t+1$ belongs to C_1 with probability 0.3. Thus the expected decrease(up to $o(1/n)$ terms) in the value of F at time $t+1$ is $0.3 \times \frac{H(t)}{0.3(n-t)} = \frac{H(t)}{n-t}$.

The differential equation follows by applying Kurtz's theorem by viewing $\vec{x}$ as a vector that maintains the current state of the solution from which the values of all the variables defined earlier can be computed. □

Lemma 16. *For all* $t \in [0, n - n^{0.9}]$ *such that* $H(t)$ *is non-negative,*

$$E\left[H(t+1) - H(t)\right] \approx -\frac{H(t)}{n-t} - 0.4\left(1 - \frac{K(t)}{0.4(n-t)}\right) - \left(\frac{K(t)}{n-t}\right)\left(\frac{SizeH(t)}{n - Last(t)}\right)$$

In the limit the above random process is closely approximated by the trajectory of the following differential equation.

$$\frac{dh}{dt} \approx -\frac{h}{n-t} - 0.4\left(1 - \frac{k}{0.4(n-t)}\right) - \left(\frac{k}{n-t}\right)\left(\frac{s}{n-\ell}\right)$$

Proof. The first term on the right hand side of the lemma follows from the proof of lemma 15. The vertex arriving at time $t+1$ is from C_2 with probability 0.4. Two cases may arise depending on whether its unique neighbor in R_2 is available or already matched.

If its unique neighbor in R_2 is already matched and $H(t)$ is greater than 0, then the vertex arriving at $t+1$ would match with the highest available vertex in R_1. The

number of vertices belonging to C_2 that are yet to arrive by time $t+1$ is tightly concentrated around $0.4(n-t)$, and of these $0.4(n-t) - K(t)$ are such that their unique neighbor in R_2 is already matched. These vertices have no choice but to get matched to a vertex in R_1. This happens with probability $0.4\left(1 - \frac{K(t)}{0.4(n-t)}\right)$.

The third term considers the case when the arriving vertex belongs to C_2 and its unique neighbor in R_2 is still available. In this case the arriving vertex would match to a vertex in R_1 only if its unique neighbor in R_2 is below the highest surviving vertex from R_1. Next, we find the probability of this event.

By conditional probability,

$$\begin{aligned}
&\Pr[\ v^* \text{ is unmatched at } t, v \text{ has not arrived by } t, v^* \text{ is below SizeH(t) }] \\
&\quad = \Pr[\ v^* \text{ is unmatched at } t, v \text{ has not arrived by } t\ ] \\
&\quad \times\ \Pr[\ v^* \text{ is below SizeH(t)} \mid v^* \text{ is unmatched by } t, v \text{ has not arrived by } t\ ]
\end{aligned} \tag{15}$$

For any vertex $v \in C_2$ that is yet to arrive by time t, let v^* be its unique neighbor in R_2. With high probability(using Chernoff bounds and collecting the bad events in the $o(1/n)$ term) there are $0.4(n-t)$ such choices of v. Hence, the probability that v^* is still unmatched at time t and v is yet to arrive is $\frac{K(t)}{0.4(n-t)}$.

To calculate the second probability term in equation (15) we note that for this event to happen v^* must be below $Last(t)$. Thus the desired probability is given by $\frac{SizeH(t)}{n-Last(t)}$.

Combining the above two arguments and using equation (15) for the case when arriving vertex belongs to C_2 and its unique neighbor in R_2 is still available, the probability of matching to a vertex in R_1 is given by $\left(\frac{K(t)}{n-t}\right)\left(\frac{SizeH(t)}{n-Last(t)}\right)$. The differential equation follows by applying Kurtz's theorem. □

In the next Lemma we give a difference equation that describes the expected behavior of K.

Lemma 17. *For all* $t \in [0, n - n^{0.9}]$

$$E\left[K(t+1) - K(t)\right] \approx -\frac{K(t)}{n-t} - 0.3\left(\frac{K(t)}{J(t)}\right)\left(\frac{n - Last(t)}{n}\right)$$

In the limit the above random process is closely approximated by the trajectory of the following differential equation.

$$\frac{dk}{dt} \approx -\frac{k}{n-t} - 0.3\left(\frac{k}{j}\right)\left(\frac{n-\ell}{n}\right)$$

Proof. We consider two cases based on the type of incoming vertex. If the incoming vertex belongs to C_2 then K would reduce by 1 only if the unique neighbor of this vertex in R_1 is still unmatched. This happens with probability $\frac{K(t)}{0.4(n-t)}$, ignoring $o(1/n)$ terms. The arriving vertex is from C_2 with probability 0.4. This yields the first term.

The second term corresponds to the case when the arriving vertex is from the set C_3. This happens with probability 0.3. Let v be the arriving vertex and v^* be its unique neighbor in R_3. Let u^* be the vertex that v gets matched to. We wish to find the probability of the event that u^* belongs $K(t)$.

The following equations can be derived using conditional probability and based on our definitions stated earlier.

$$\begin{aligned}
\Pr\left[K(t) \text{ reduces by } 1 \mid v \in C_3\right] &= \Pr\left[u^* \in K(t)\right] \\
&= \Pr\left[u^* \in R_2\right] \cdot \Pr\left[u^* \in K(t) \mid u^* \in R_2\right] \\
&= \left(\frac{n - Last(t)}{n}\right) \cdot \left(\frac{K(t)}{J(t)}\right)
\end{aligned}$$

The lemma follows from the above arguments. The differential equation follows by applying Kurtz's theorem. □

In the following two lemmas we will argue about the expected behavior of J and $SizeH$ as a function of t.

Lemma 18. *For all* $t \in [0, n - n^{0.9}]$,

$$E\left[J(t+1) - J(t)\right] \approx -0.3\left(\frac{n - Last(t)}{n}\right) - \left(\frac{K(t)}{n-t}\right)\left(\frac{SizeH(t)}{n - Last(t)}\right)$$

In the limit the above random process is closely approximated by the trajectory of the following differential equation.

$$\frac{dj}{dt} \approx -0.3\left(\frac{n-\ell}{n}\right) - \left(\frac{k}{n-t}\right)\left(\frac{s}{n-\ell}\right)$$

Proof. Two cases may arise depending on the type of the arriving vertex. The two terms correspond to these cases. Let the arriving vertex be $v \in C_3$, and let v^* be its unique neighbor in R_3. This happens with probability 0.3. v will match to a vertex in R_2 if v^* is below the highest unmatched vertex in R_2 i.e. $\rho^{-1}(v^*) \in [Last(t), n]$. Thus, the total probability of a match in R_2 in this case is $0.3\left(\frac{n-Last(t)}{n}\right)$.

Now let us consider the case when the arriving vertex is from C_2. This happens with probability 0.4. Clearly the case of interest is when the unique neighbor of v in R_2, say v^* is still unmatched at t. As before this happens with probability $\frac{K(t)}{0.4(n-t)}$.

Using conditional probability we have,

$$\begin{aligned}
\Pr\left[v \text{ matches } v^*\right] &= 0.4 \times \Pr\left[v \text{ matches } v^* | v^* \text{ is unmatched at } t\right] \\
&\quad \times \Pr\left[v^* \text{ is unmatched at } t\right] \\
&= \left(\frac{K(t)}{n-t}\right) \times \Pr\left[v \text{ matches } v^* | v^* \text{ is unmatched at } t\right] \\
&= \left(\frac{K(t)}{n-t}\right) \times \left(\frac{SizeH(t)}{n - Last(t)}\right)
\end{aligned}$$

The last line follows from the observation that v^* is unmatched at time t if $\rho^{-1}(v^*) \in (Last(t), n]$ and v will match v^* only if it is placed higher than the first unmatched vertex in R_1 i.e. $\rho^{-1}(v^*) \in [0, SizeH(t))$. The differential equation follows by applying Kurtz's theorem. □

Lemma 19. *For all* $t \in [0, n - n^{0.9}]$ *with high probability,*

$$SizeH(t) \approx \frac{nH(t)}{0.3(n-t)}$$

Proof. H is uniformly distributed in the range
$[n - SizeH(t), n]$. For any vertex in this range the probability that it belongs to R_2 and is its unique neighbor in C_1 has still not arrived is $\frac{0.3(n-t)}{n}$ (ignoring lower order terms). Such vertices belong to $H(t)$. Thus we have,

$$\frac{H(t)}{SizeH(t)} \approx \frac{0.3(n-t)}{n}$$

This yields the desired result. □

The differential equations described in lemmas 15, 16, 17, 18, representing the expected behavior of the variables can be solved analytically under the proper boundary conditions to give $F(n)/n \approx 0.197$ and $J(n)/n \approx 0.076$. Thus the factor for the algorithm is $1 - (0.197 + 0.076) = 0.727$.

2.5 Conclusion and Open Problems

In this chapter we studied the online bipartite matching problem when the arriving vertices are drawn from a fixed but unknown distribution. We demonstrated that the RANKING algorithm described [37] in the context of adversarial model attains a factor of 0.656 in the unknown distribution model. Our proof was based on analyzing the stricter random order arrival model and exploited certain symmetry properties that are absent in the adversarial model. We also presented a family of graphs for which RANKING does no better than 0.727.

While our results show that the random order arrival model is strictly more powerful than the adversarial model it is still unknown whether we can establish a similar separation between the random order arrival model and the more general unknown distribution model. We leave this question as an open problem.

CHAPTER III

MATCHING IN THE OBLIVIOUS QUERY COMMIT MODEL

3.1 Introduction

Several efficient combinatorial algorithms are known for the maximum matching problem [17, 48] for general graphs; however these algorithms require complete information about the underlying graph and rely on building global combinatorial structures such as *blossoms*. This may not be feasible in some settings where the underlying graph may be hidden and the algorithm is constrained to make decisions based solely on locally available information. For example consider the kidney exchange problem - Often patients with a kidney disease have a family member who is willing to donate his/her kidney. Unfortunately, sometimes these donors may be blood-type incompatible with the patient. To solve this problem, a kidney exchange is performed in which patients swap their incompatible donors to get a compatible donor. Owing to the cost involved in medical tests, incentive issues, and due to ethical concerns, it is desired that an exchange is performed whenever the test indicates that the exchange is possible. To increase the efficiency of the kidney exchange program, it is important to match the maximum number of compatible patient-donor pairs. Refer to [56, 57, 58] for further background on this problem.

Formally this problem can be modeled as a maximum matching problem in a general non-bipartite graph where each patient-donor pair represents a node of the graph and an edge between two nodes indicate if an exchange is possible. For every pair $(u, v) \in V \times V$ we are *not* told a priori whether there is an edge connecting these vertices, until we *probe/scan* this pair. If we scan a pair of vertices and find that

there is an edge connecting them we are constrained to *pick* this edge and in this case both u and v are removed from the graph. However, if we find that u and v are not connected by an edge, they continue to be available to be matched in the future. The goal is to maximize the number of vertices that get matched. We refer to the above as the *oblivious query commit model* since we are oblivious of the underlying graph yet are required to commit to choosing an edge as soon as it is discovered.

It is easy to see that any algorithm that probes all permissible edges is a $1/2$ approximation since we are guaranteed to pick a maximal matching which is at least half as big as the largest matching in the underlying graph. One can also construct examples to show that no deterministic algorithm can do strictly better than $1/2$ on all instances. Thus our only hope is to seek a randomized algorithm that beats the barrier of $1/2$ in expectation.

Arguably the most natural randomized algorithm is the one that tests all vertex pairs in a random order. However this algorithm can have an approximation factor as bad as $1/2 + o(1)$. For example consider a graph over n vertices, such that half the vertices form a clique, while for the other half of the vertices, each one of them is adjacent to a unique vertex in the clique. The above algorithm would only attain a factor of $1/2 + \log n/n$ for such graphs.

Hence, to get a better approximation factor, one could possibly use correlated randomness (as opposed to sampling edges at random in every step) to determine the order in which the edges should be scanned. This idea was first used by Aronson, Dyer, Frieze, and Suen in [3], where they gave the following randomized algorithm for the maximum matching problem - Pick a vertex at random and match it to one of its unmatched neighbors uniformly at random. They showed that this algorithm does marginally better than 0.5 and attains a factor of 0.50000025. Note that this algorithm can be simulated in the query commit model by first picking a random vertex u and then scanning pairs (u, v) for all unmatched vertices $v \in V$ in a random

order, until u gets *matched* or it is established that u cannot be matched with the remaining unmatched vertices.

3.1.1 Our Results

From the above discussion it is clear that choosing the correct correlated randomization to determine the order in which the edges are scanned is critical to get a good approximation factor. In this work, we propose and analyze the following randomized algorithm (referred to as SHUFFLE): Shuffle the vertices according to a uniformly random permutation ρ and iterate through them one at a time. If the current vertex is already matched then ignore it else scan edges incident to it in the order dictated by ρ until it gets matched or there are no vertices left. Then proceed to the next vertex.

Note that our algorithm differs from the algorithm given by [3] in the sense that after picking a random vertex u when we scan all the edges (u, v) for $v \in V$ in a *specific order* that is given by the randomization used to pick the vertex u itself, instead of scanning them in a random order, as suggested by [3].
We show the following results:

(a) SHUFFLE attains a factor of 0.56 for the oblivious query commit problem, and thus improves upon the 0.50000025-factor algorithm given by Aronson, Dyer, Frieze, and Suen in [3].

(b) There exists a family of graphs for which SHUFFLE attains a factor no better than 0.727.

(c) We show that no randomized algorithm can attain a factor better than 0.7916 for the oblivious query commit problem.

(d) We also show a tighter bound for a large class of algorithms called vertex iterative (VI) algorithms. Both SHUFFLE and the algorithm in [3] fall in this

class. A VI algorithm considers the vertices one at a time and for every vertex probes edges incident on it until it gets matched. A vertex may also choose to "give-up" in which case it plays no further part in the algorithm and we move on to the next vertex. The order in which the vertices are scanned, and the sequence in which we scan the edges may be determined adaptively i.e. it can depend on the past outcomes. For this general class of algorithms we show that no randomized algorithm can attain a factor better than 0.75.

3.1.2 Related Models and Results

Local Algorithms for Matching: As mentioned earlier, the query commit model essentially limits us to implement myopic Greedy-like algorithms. Furthermore, since the greedy algorithms are easy to implement and can be easily adapted to different environments, for instance distributed setting, they are quite prevalent in practice. As a result, there is some interest in designing better randomized greedy algorithms that outperform the 0.5 approximation of a deterministic greedy algorithm. In [16] the authors studied the greedy algorithm where an edge is picked uniformly at random among the unmatched vertices. They showed that for general graphs it doesn't give any improvement asymptotically, and for sparse graphs it significantly improves the approximation factor. Later, [3] gave a different randomized greedy algorithm that achieves a factor 0.50000025 for general graphs.

In other related work, designing fast algorithms for finding approximate maximum matchings has also received considerable attention recently [53] [66] [15]. However all these algorithms explicitly exploit the edge structure of the graph and are not applicable in our setting.

Connection to the Ranking **algorithm:** The Shuffle algorithm presented in this chapter is closely related to the Ranking algorithm defined in Section 2. In fact in Lemma 32 we show that these two algorithms are identical for bipartite graphs.

Thus both the upper and lower bounds from Chapter 2 carry over to the analysis of SHUFFLE in the context of bipartite graphs i.e. SHUFFLE attains a factor of at least 0.656 for bipartite graphs and there exits a family of (bipartite) graphs for which it does no better than 0.727.

3.1.3 Technical Contributions

Conquering Non-monotonicity: A vast body of prior work [1, 6, 24, 36, 37, 44] on online allocation problems has relied on a crucial property called *monotonicity*. The property is formally defined in Section 3.2.2, but intuitively it states that adding new vertices can not cause one of its previously matched neighbors to get unmatched. This property holds for bipartite graphs for most natural algorithms like RANKING, but unfortunately as shown in the simple example in Figure 3, it does not hold for all graphs in general. Observe that at the heart of the example is an odd cycle which cannot happen in bipartite graphs.

Thus the lack of monotonicity is a massive blow in the quest for an algorithm for the query commit problem. We overcome this obstacle by defining another property called *stability*. This property states that adding a vertex can only alter the matching found by SHUFFLE by at most a single augmenting path. Using this observation we show that every time a vertex does not conform to the monotonicity property in SHUFFLE it results in a certain types of *good-events*. Finally we bound the approximation factor for the algorithm in terms of the number of these good events.

Strongly Factor Revealing Family of Linear Programs: In our analysis we prove several combinatorial lemmas to lower bound the performance of SHUFFLE. We then coalesce them into a large linear program parameterized by the size of the input, that lower bounds the approximation factor. However the linear program is quite complicated and not amenable to traditional means of analysis that are used to study factor revealing linear programs (refer to [47][42][35][45]). Our analysis is based

on a technique recently introduced in [44]. In this method we use these parameterized LPs to derive a new family of LPs (strongly factor revealing family) each of which lower bounds the approximation factor SHUFFLE for *any* given input size. Thus solving any large enough instance from this family serves to bound the performance of SHUFFLE.

Upper Bound using Yao's Lemma [69]: We consider a general class of algorithms called vertex iterative algorithms and prove a lower bound on the performance of any such algorithm using Yao's Lemma [69]. This entails finding the best deterministic algorithm for an appropriately chosen distribution over the inputs. Unfortunately it turns out to be quite difficult to characterize the optimal deterministic algorithm for the distribution that we consider in our proof. Instead we define a class of fictitious deterministic algorithms called *revealing-algorithms* and show that they perform at least as well as any deterministic vertex iterative algorithm. Then to analyze the optimal revealing-algorithm for our distribution we define another randomized revealing algorithm and argue that its expected performance is at least as good as the best deterministic revealing algorithm. Finally we bound the performance of this randomized algorithm to complete our proof.

3.2 Preliminaries

3.2.1 Problem Statement

Let $G(V, E)$ be a non-bipartite graph where $|V| = n$. We wish to analyze the performance of the following algorithm for the query commit problem.

SHUFFLE **Algorithm**

1. Choose a uniformly random permutation ρ of the vertices.
2. Apply permutation ρ to V - thereby assigning each vertex a priority or rank.
3. Process the vertices one at a time in the increasing order of the rank. If the vertex under consideration, say vertex u, is already matched then ignore it, else

scan edges incident to it from the unmatched vertices in an increasing order of their rank until vertex u gets matched. Then proceed to the next vertex.

We will use u, v to denote the vertices of V and $s, t \in [n]$, to index the vertices in V. Let ALG(G) be the expected size of the matching returned by the above algorithm and let OPT(G) be the largest matching in G. We say that SHUFFLE attains a factor of α if $ALG(G) \geq \alpha|OPT(G)|$ for every graph G.

Let us now recall some notation developed in Chapter 2 that will be useful in our analysis. Define Ω_V to be the set of all permutations of V. For any permutation ρ of the vertices, we will use $\rho(t)$ to denote the vertex at the t^{th} position in ρ and $\rho^{-1}(u)$ to denote the position of the vertex u in ρ. For vertices u, v we say u is above v in ρ if $\rho^{-1}(u)$ is less than $\rho^{-1}(v)$. We can similarly define the notion of a vertex u being below another vertex v. For any permutation $\rho \in \Omega_V$ let $\rho[u \rightsquigarrow s]$ denote the permutation obtained by moving u to position s keeping the order of the other vertices unchanged.

For any $t \in [n]$ define x_t to be the probability that the vertex $\rho(t) \in V$ at position t gets matched in SHUFFLE, where the probability is taken over the random choices of ρ. We will use $Shuffle(\rho)$ to denote an invocation of SHUFFLE with ρ as the permutation chosen in the first step. In general for any algorithm $\mathcal{A}$ and graph G we will use $\mathcal{A}(G)$ to denote the performance of $\mathcal{A}$ on G.

3.2.2 Properties of the SHUFFLE Algorithm

In this section we will introduce terminology that would be central to our analysis.

Monotonicity Property: Consider $u = \rho(t)$ that is unmatched by $Shuffle(\rho)$. Suppose v is another vertex that is adjacent to u, i.e. $uv \in E$, then v is surely matched to a vertex above u. The monotonicity property strengthens this observation, by allowing us to translate u to any of the n positions in ρ. Formally, we say a vertex u satisfies the monotonicity property with respect to v for a given permutation ρ if,

1. $uv \in E$
2. u is unmatched in $Shuffle(\rho)$
3. v is matched in $Shuffle(\rho)$
4. v is matched above position t in $Shuffle(\rho[u \rightsquigarrow s])$ for every $s \in [n]$

Non-monotone Event: Unfortunately some vertices may not satisfy the monotonicity property for all choices of s (in the fourth point above). This prompts us to define a non-monotone event. For a given permutation ρ, a vertex $u = \rho(t)$ is involved in a non-monotone event with respect to another vertex v and position s if the following conditions are met.

1. $uv \in E$
2. u is unmatched in $Shuffle(\rho)$
3. v is matched in $Shuffle(\rho)$
4. v is matched below position t or unmatched in $Shuffle(\rho[u \rightsquigarrow s])$

That is, promoting u in ρ causes one of its previously matched neighbors to match lower or to get unmatched. We use $\Gamma^v(s, t, \rho)$ to be the indicator variable for this event. For illustration, consider the example shown in Figure 3.

The matched edges are shown in bold while the other edges are indicated by dashed lines. Here we consider two permutations $\rho = [x, y, v, w, u]$ (Figure 3a) and $\rho' = [u, x, y, v, w]$ (Figure 3b). Note that moving u to the start of the permutation ρ causes one of its previously matched neighbors v to get unmatched in $Shuffle(\rho')$. In this case we say u has *generated* a non-monotone event by moving to position 1, i.e., $\Gamma^v(1, 5, \rho) = 1$ for the above example.

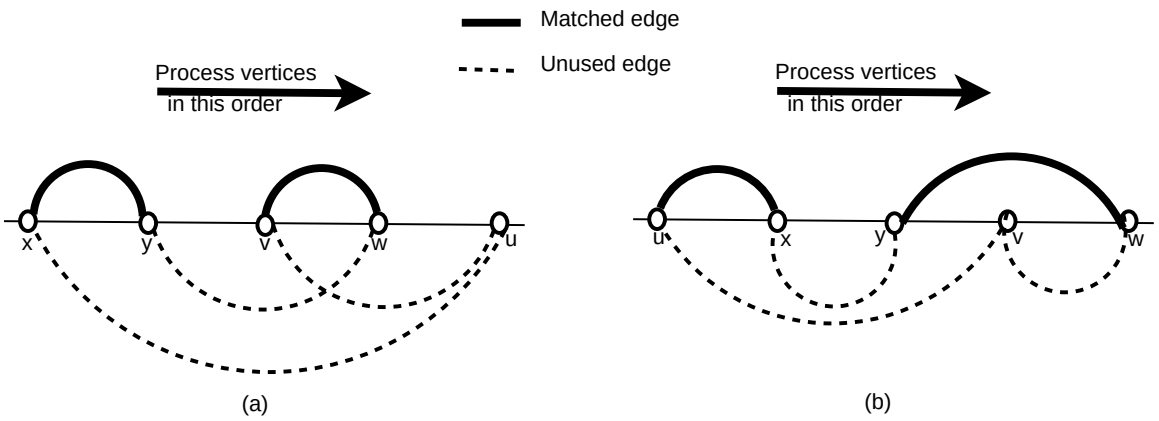

Figure 3: Non-monotone event generated by translating u

Observe that $\{x, y, w, v, u\}$ form a cycle with odd number of vertices. In fact every non-monotone event can be associated with an odd cycle. This explains why such events do not occur when the underlying graph is bipartite.

Stability Property: At a high level this property states that for any $\rho \in \Omega_V$ the execution of $Shuffle(\rho)$ does not change dramatically by altering the position of a single vertex in ρ. For any graph $H(V, E)$ consider $\rho \in \Omega_V$. For an arbitrary vertex $u \in V$ let $\rho' = \rho[u \rightsquigarrow s]$. Consider the execution of $Shuffle$ for graph H and for any $v \in V$, define $M(v, \rho)$ to be the set of matched edges when v was considered in $Shuffle(\rho)$. Similarly let $M(v, \rho')$ be the set of matched edges when v was considered by $Shuffle(\rho')$. The stability property is summarized by the following claim that can be proved by induction.

Claim 1. *For all $v \in V - u$, the symmetric difference of $M(v, \rho)$ and $M(v, \rho')$ has at most one component, i.e., it is either empty or a path or a cycle.*

Good-Events: We will use a notion of good-events to quantify our algorithm's improvement over the greedy algorithm that matches exactly half the vertices in the worst case. A pair of vertices that are matched to each other in the optimal solution would correspond to some good-events if they both are also matched (even if not

to each other) in an execution of the algorithm. We will attribute any such pair of vertices to two types of good events that are defined below.

- *Type 1 good-event*: For any given permutation $\rho \in \Omega_V$ under consideration, we say a good-event of type 1 happens at position t if both $\rho(t)$ and its partner(if any) in OPT(G) are matched by $Shuffle(\rho)$. We will use $Good_1(t, \rho)$ as the indicator variable for this event.

- *Type 2 good-event*: The second type of good event is defined in a slightly indirect manner. For a permutation $\rho \in \Omega_V$ we say a good event of type 2 happens at position t if the following conditions are met

  1. $u = \rho(t)$ is matched in $Shuffle(\rho)$. Let w be the vertex that it is matched to.
  2. If w^* is the partner (if any) of w in OPT(G), then w^* is matched in $Shuffle(\rho)$.
  3. Consider ρ' that is produced by deleting u from ρ and keeping the relative order of all other vertices the same. Then, both w and w^* are matched to *each other* in $Shuffle(\rho')$.

From the above definitions it is clear that type 2 good events are a subset of type 1 good events. Also note that if OPT(G) matches w to w^*, and both the vertices are matched in our algorithm for a given permutation, then we generate two good events of type 1, one at the position of w and other at the position of w^*. On the other hand, one can show that we will generate at most one good event of type 2, which will be at the position of the vertex u, where u is matched to the vertex that is first to be matched among w and w^*.

3.3 Analysis of the SHUFFLE Algorithm

We divide the analysis into two parts. In the first part, we use combinatorial arguments to set up equations relating the variables defined in Section 3.2.2. In the second part of the analysis we use these equations to give a lower bound on the performance of SHUFFLE through a strongly factor revealing LP.

Intuitively, the input space of graphs has two types - graphs that have *few* non-monotone events(such as bipartite graphs) and those that have *large* number of non-monotone events. In the first case one can leverage the monotonicity property to show that the algorithm has a good approximation factor. The analysis for the second type of graphs is little more involved and relies on the observation that if there are a large number of good-events then we get a good approximation factor. To show that there are non-negligible number of good-events we prove a structural lemma - that forms the core of our arguments - to lower bounds the number of good-events in terms of the number of non-monotone events. To the best of our knowledge, this is the first work that relates good-events to non-monotone events for the matching problems (and, in general, allocation problems), and uses it to show a non-trivial improvement over the standard greedy algorithm.

3.3.1 Combinatorial Arguments

We begin by proving that the worst examples for SHUFFLE must have a perfect matching.

Lemma 20. *We may assume without loss of generality that the graphs for which* SHUFFLE *attains the worst factor has a perfect matching.*

Proof. Let $H(V, E)$ be the graph for which SHUFFLE attains the worst factor. Suppose H does not have a perfect matching. Let v be a vertex in H that is not matched in the optimal matching and let $H' = H/v$. Thus OPT(H) = OPT(H'). We will prove

that ALG(H') $\leq$ ALG(H), thereby showing that H' is also a worst-case example for SHUFFLE.

Consider $\rho \in \Omega_V$ and let ρ' be the permutation produced by deleting v from ρ. Let M be the matching returned by $Shuffle(\rho)$ and let M' be the matching found by $Shuffle(\rho')$. We will show that $|M| \geq |M'|$.

Let us consider two cases. If v is unmatched in $Shuffle(\rho)$ then $|M| = |M'|$. On the other hand if v is matched in M then by Claim 1 the symmetric difference of M and M' is a path or a cycle containing v. If the symmetric difference is a cycle then $|M| = |M'|$ and we are done. On the other hand if it is a path, say P, then at most one vertex in P can be unmatched in each of the matchings M and M'. This together with the fact that v only belongs to H and is matched in M implies that $|M| \geq |M'|$.

Since removing v does not alter the size of the optimal solution, the performance of SHUFFLE on H' is no better than its performance on H. We can iteratively remove all vertices that are unmatched in OPT(H) and by similar arguments as above, show that this can only degrade the performance of SHUFFLE. □

For the rest of this chapter we will assume that the given graph G has a perfect matching. Furthermore, for any $u \in V$, we will use u^* to denote its match in OPT(G) we will refer to u^* (resp. u) as the *partner* of u (resp. u^*) with respect to OPT(G). Also for brevity we will use $\Gamma(s, t, \rho)$ to denote $\Gamma^{u^*}(s, t, \rho)$ where $u = \rho(t)$. Thus $\Gamma(s, t, \rho)$ refers to the non-monotone event where translating an unmatched vertex $u = \rho(t)$ to position s in ρ causes its partner u^* (which was previously matched above t) to get unmatched or to match below t.

The following observation follows easily from the definitions of x_t.

Observation 1. $\forall t \in [n]$: $\quad x_t \geq x_{t+1}$

The next lemma is similar in spirit to Observation 1 above.

Lemma 21. *For any $s, t \in [n]$ such that $s < t$,*

$$\sum_{\rho \in \Omega_V} \Gamma(s, t, \rho) \leq \sum_{\rho \in \Omega_V} \Gamma(s, t+1, \rho)$$

Proof. Consider $\rho \in \Omega_V$ such that we generate a non-monotone event $\Gamma(s, t, \rho)$ by promoting $u = \rho(t)$ to position s. Let $\rho' = \rho[u \rightsquigarrow t+1]$. Since u is unmatched at position t in ρ it will remain unmatched at position $t+1$ in $Shuffle(\rho')$. Also translating u from position $t+1$ to s in ρ' will generate a non-monotone event $\Gamma(s, t+1, \rho')$. Thus we can set up an injective map from non-montone events of the form $\Gamma(s, t, .)$ to non-monotone events of the form $\Gamma(s, t+1, .)$. This suffices to prove the above claim. □

The following lemma relates x_t and $\Gamma(s, t, \rho)$.

Lemma 22. *For every $s, t \in [n]$,*

$$|\Omega_V|(1 - x_t) \geq \sum_{\rho \in \Omega_V} \Gamma(s, t, \rho)$$

Proof. The total number of times the vertex at position t remains unmatched is given by $|\Omega_V|(1 - x_t)$. Since a necessary requirement(by property 2 in the definition) of a having a nonmonotone event $\Gamma(s, t, \rho)$ is that $\rho(t)$ is unmatched in $Shuffle(\rho)$, the number of such non-monotone events is at most $|\Omega_V|(1 - x_t)$. □

Lemma 23 follows from similar arguments as above.

Lemma 23. *For every $t \in [n]$,*

1. $|\Omega_V| x_t \geq \sum_{\rho \in \Omega_V} Good_1(t, \rho)$
2. $|\Omega_V| x_t \geq \sum_{\rho \in \Omega_V} Good_2(t, \rho)$

We will now present three bounds on ALG(G). The first follows trivially from the definition of x_t.

Observation 2.

$$ALG(G) = \sum_{t \in [n]} x_t$$

The next two bounds are slightly more involved and make use of the two types of good-events defined in Section 3.2.2. These are proved in Lemma 24 and 25 and illustrate the role of good-events in attaining a factor better than 0.5.

Lemma 24.

$$ALG(G) = \frac{n}{2} + \frac{\sum_t \sum_{\rho \in \Omega_V} Good_1(t, \rho)}{2|\Omega_V|}$$

Proof. By Lemma 20 we may assume, without loss of generality, that the underlying graph has a perfect matching. Any algorithm(including SHUFFLE) that returns a maximal matching would surely match at least one of w and w^* for any vertex $w \in V$ and therefore matches at least $n/2$ vertices.

To find the total number of vertices that get matched we also need to add the number of pairs (w, w^*) such that both w and w^* are matched. Since each such pair contributes twice to the summation $\sum_t \sum_{\rho \in \Omega_V} Good_1(t, \rho)$, the expected number of vertices matched by SHUFFLE is given by $\frac{n}{2} + \frac{\sum_t \sum_{\rho \in \Omega_V} Good_1(t,\rho)}{2|\Omega_V|}$. □

Lemma 25.

$$ALG(G) \geq \frac{n}{2} + \frac{\sum_t \sum_{\rho \in \Omega_V} Good_2(t, \rho)}{|\Omega_V|}$$

Proof. As before by Lemma 20 we may assume, that the underlying graph has a perfect matching. Thus any greedy algorithm(including SHUFFLE) matches at least $n/2$ vertices. To bound the total number of vertices that get matched we also need to add the number of pairs (w, w^*) such that both w and w^* are matched. We do this by constructing a map from good-events of type 2 into the set of such pairs.

Consider a good-event $Good_2(t, \rho)$ and let $u = \rho(t)$. Let w be the vertex that u is matched to in $Shuffle(\rho)$. By definition of a good-event of type 2, w^* is also

matched in $Shuffle(\rho)$. Thus each type 2 good-event maps to a pair (w, w^*) such that both w and w^* are matched. Also observe that this map is one-one, i.e. given a pair (w, w^*) that lies in the range of this map there is exactly one good-event of type 2 that could be its preimage. This is because if we define $\hat{w} \in \{w, w^*\}$ to be the first among w and w^* to get matched in $Shuffle(\rho)$ then the neighbor of $\hat{w}$ in $Shuffle(\rho)$ uniquely defines the vertex u that if deleted would cause w and w^* to be matched. □

The following lemma illustrates the connection between nonmonotone events and good-events. It is the pivotal technical lemma in the analysis. Finally in Lemma 27 we relate the number of non-monotone events bounds to the total number of miss events.

Lemma 26. *For every $t \in [n]$,*

$$\sum_{\rho \in \Omega_V} \Gamma(t, n, \rho) \leq \sum_{\rho \in \Omega_V} Good_2(t, \rho) + \frac{n \sum_{s \leq t} \sum_{\rho \in \Omega_V} Good_1(s, \rho)}{t}$$

Proof. The proof of this Lemma relies on setting up two maps that associate good-events (of type 1 or 2) with every non-monotone event. Let $\rho \in \Omega_V$ such that $\Gamma(t, n, \rho) = 1$. Let $u = \rho(n)$ and $\rho' = \rho[u \rightsquigarrow t]$. Clearly u is matched by $Shuffle(\rho')$, let $w \neq u^*$ be the vertex that it is matched to. Let w^* be the partner for w in $OPT(G)$. We consider the following two cases.

Case 1: Both w and w^* were matched to each other in $Shuffle(\rho)$.

Since we have a non-monotone event when we move u to position t, w^* should surely be matched in $Shuffle(\rho')$. Thus in this case moving u to position t in ρ generates a good-event of type 2 at position t in $Shuffle(\rho')$. Thus in this case, every non-monotone event $\Gamma(t, n, \rho)$ generates exactly one good event of type 2. This is represented by the first term on the right-hand side of the above inequality.

Case 2: w and w^* were not matched to each other in $Shuffle(\rho)$.

Suppose we move w^* around in ρ', i.e. consider $\rho'' = \rho'[w^* \rightsquigarrow s]$ for $s \in [n]$ then we have the following two claims. The first claim says that w would always stay matched, while the second claim says that for at least(the top) t positions of w^* both w and w^* will get matched. We defer the proofs to Section 3.5.

Claim 2. *w is matched in $Shuffle(\rho'[w^* \rightsquigarrow s])$ for all choices of $s \in [n]$.*

Claim 3. *w^* is matched by $Shuffle(\rho'[w^* \rightsquigarrow s])$ for all choices of $s \leq t$, $s \in [n]$.*

By Claim 2 w stays matched for all positions of w^*. Of the n positions that w^* can take, it too gets matched for at least t positions. Thus we generate at least t good-events of type 1 (for the top t positions of w^*) by moving w^* over ρ' from every non-monotone event $\Gamma(t, n, \rho)$.

However, we may over count while accounting for good-events since the same type 1 good-event can be generated from multiple nonmonotone events. Let us fix t for which we are writing the equation. Consider a good-event $Good_1(s, \hat{\rho})$ generated as defined above. Let $u = \hat{\rho}(t)$ and $w^* = \hat{\rho}(s)$. Let us count the number of nonmonotone events $\Gamma(t, n, \rho)$(characterized by permutation ρ) that may generate this good-event. The positions of all vertices except w^* and u are predetermined by their positions in $\hat{\rho}$. Now u should surely be at position n in ρ, but the position of w^* remains ambiguous. It may be at any of the n positions in ρ. Thus in the worst case there may be n ways to choose ρ such that the nonmonotone event $\Gamma(t, n, \rho)$ generates the good-event $Good_1(s, \hat{\rho})$. Thus every non-monotone event generates t good-events of type 1 and each such good-event may be generated by as many as n non-monotone events. Substituting the variables gives us the second term in the above inequality. □

Lemma 27. *For every $t \in [n]$,*

$$n(1 - x_t) \leq \sum_{s \leq t} x_s + \frac{\sum_{s \leq t} \sum_{\rho \in \Omega_V} \Gamma(s, t, \rho)}{|\Omega_V|}$$

Proof. We will prove this lemma by defining a function that maps a miss event at position t to n events that are either match events above t or to a non-monotone events above t. Consider $\rho \in \Omega_V$ such that $u = \rho(t)$ is unmatched. Consider $Shuffle(\rho[u \rightsquigarrow s])$ for all values of $s \in [n]$ and suppose we analyse what happens to u^* in each of these events. It will either continue to be matched above position t or it may get matched below position t(or get unmatched). In the former case we generate a match event above t, while in the latter we generate a non-monotone event at s, $\Gamma(s, t, \rho)$. One can check that the set of events generated for distinct miss events are disjoint.

There are $|\Omega_V|(1 - x_t)$ miss events in all and the total number of match events above t is given by $|\Omega_V| \sum_{s \leq t} x_s$. Using the above map we have

$$n|\Omega_V|(1 - x_t) \leq |\Omega_V| \sum_{s \leq t} x_s + \sum_{s \leq t} \sum_{\rho \in \Omega_V} \Gamma(s, t, \rho)$$

Dividing throughout by $|\Omega_V|$ proves the lemma. □

Remark 1. *A curious reader may ask about the case when there are no nonmonotone events, which would render Lemma 26 useless. However, in this case Lemma 27 can be restated as* $1 - x_t \leq \sum_{s \leq t} x_s$. *Through a simple factor revealing LP, along with Observation 2, this suffices to show that* $ALG(G) \geq n(1 - 1/e)$. *Also recall that if we have lots of non-monotone events then Lemmas 24, 26, 27 ensure that* Shuffle *attains a factor better than* 0.5. *The remainder of the analysis is focused on balancing these two effects.*

In the next section we will demonstrate how the above structural Lemmas can be used to show that Shuffle attains a factor of at least 0.560.

3.3.2 Strongly Factor Revealing Linear Program

Let us define g_t to be the probability of a type 1 good event at position t, i.e. $g_t = \frac{\sum_{\rho \in \Omega_V} Good_1(t,\rho)}{|\Omega_V|}$. Similarly define $h_t = \frac{\sum_{\rho \in \Omega_V} Good_2(t,\rho)}{|\Omega_V|}$ and $\gamma_{s,t} = \frac{\sum_{\rho \in \Omega_V} \Gamma(s,t,\rho)}{|\Omega_V|}$. In

light of the above definitions the results of Section 3.3.1 can be reformulated as the following linear program.

$$\text{LP(n)} : \text{minimize} \quad \frac{ALG}{n}$$

Subject to

$$ALG = \sum_{t \in [n]} x_t \qquad \text{Observation 2} \quad (16)$$

$$ALG = \frac{n}{2} + \frac{1}{2}\sum_{t} g_t \qquad \text{Lemma 24} \quad (17)$$

$$ALG \geq \frac{n}{2} + \sum_{t} h_t \qquad \text{Lemma 25} \quad (18)$$

$$\gamma_{t,n} \leq h_t + \frac{n\sum_{s \leq t} g_s}{t} \qquad \forall t \in [n] \qquad \text{Lemma 26} \quad (19)$$

$$n(1 - x_t) \leq \sum_{s \leq t} x_s + \sum_{s \leq t} \gamma_{s,t} \qquad \forall t \in [n] \qquad \text{Lemma 27} \quad (20)$$

$$1 - x_t \geq \gamma_{s,t} \qquad \forall s, t \in [n] \qquad \text{Lemma 22} \quad (21)$$

$$x_t \geq x_{t+1} \qquad \forall t \in [n] \qquad \text{Observation 1} \quad (22)$$

$$\gamma_{s,t} \leq \gamma_{s,t+1} \qquad \forall s, t \in [n] \qquad \text{Lemma 21} \quad (23)$$

$$x_t \geq g_t \qquad \forall t \in [n] \qquad \text{Lemma 23} \quad (24)$$

$$x_t \geq h_t \qquad \forall t \in [n] \qquad \text{Lemma 23} \quad (25)$$

$$0 \leq x_t, \gamma_{s,t}, g_t, h_t \leq 1 \qquad \forall s, t \in [n] \quad (26)$$

Here we are trying to minimize the approximation factor of the algorithm subject to the constraints derived in Section 3.3.1. The following lemma follows immediately.

Lemma 28. *LP(n) lower bounds the performance(approximation factor) of* SHUFFLE *on any graph having n vertices.*

Table 2 gives the optimal value of LP(n) for different choices of the parameter n. There are two main approaches towards rigorously estimating approximation factors using factor revealing LPs. The first method relies on analytically solving the given

Table 2: Optimal Values of LP(n)

n	Optimal Value of LP(n)	Factor = LP(n)/n
20	11.124	0.5562
50	27.913	0.5582
100	55.899	0.5588
200	111.870	0.5593
300	157.879	0.5596
400	224.001	0.560

LP for an arbitrary sized input and then arguing about the optimality of the solution (refer to [47, 42]). The second technique relies on observing patterns in the dual for the given LP and using these to guess a near optimal dual for an arbitrary program (e.g. [35, 45]). This approach is usually lossy and fails to attain the optimal approximation factor. Both of these approaches are infeasible in our context since the linear program (and its dual) at hand is quite complex and not amenable to analytical study. Therefore, in order to rigorously establish that SHUFFLE does attain a factor of at least 0.560 we will employ a technique inspired by the recent work by [44].

In this technique we construct a family of progressively stronger linear programs (called a *strongly factor-revealing family*) such that the solution to any of these programs lower bounds the approximation ratio of the algorithm. Towards this end we will in fact prove that the optimal value of $LP(k)$ for any small constant k lower bounds the optimal value of $LP(n)$ for an any choice of $n >> k$. Thus our family of strongly factor-revealing linear programs is simply $\{LP(1), LP(2) \cdots LP(k)\}$. Concretely, we will show LP-OPT(k) $\leq$ LP-OPT(n), where LP-OPT(n) is the optimal value of $LP(n)$.

Lemma 29. *For any fixed constant k dividing n,*

$$LP\text{-}OPT(k) \leq LP\text{-}OPT(n)$$

Proof. Let $s^n = (ALG^n, x^n, g^n, h^n, \gamma^n)$ be the optimal solution for $LP(n)$. We will use these to construct a feasible solution $\hat{s}^k = (\widehat{ALG}^k, \hat{x}^k, \hat{g}^k, \hat{h}^k, \hat{\gamma}^k)$ for $LP(k)$ of value

LP-OPT^n thus proving LP-OPT$^k \leq$ LP-OPT^n. Let $q = n/k$. We will use $i, j \in [k]$ to index variables in $\hat{s}^k$ and s, t to index variables in s^n. Define

$$\hat{x}^k_i = \frac{\sum_{s:\lfloor s/q \rfloor = i} x^n_s}{q} \qquad \forall i \in [k] \tag{27}$$

$$\hat{g}^k_t = \frac{\sum_{s:\lfloor s/q \rfloor = i} g^n_s}{q} \qquad \forall i \in [k] \tag{28}$$

$$\hat{\gamma}^k_{i,j} = \frac{\sum_{s:\lfloor s/q \rfloor = i} \sum_{t:\lfloor t/q \rfloor = j} \gamma^n_{s,t}}{q^2} \qquad \forall i, j \in [k] \tag{29}$$

$$\widehat{ALG}^k = \frac{ALG^n}{q} \tag{30}$$

In the following 6 claims we establish that the solution defined above is a feasible solution to $LP(k)$. We defer the proofs to Section 3.5.

Claim 4. $\widehat{ALG}^k = \sum_{t \in [k]} \hat{x}^k_t$

Claim 5. $\widehat{ALG}^k = k/2 + \frac{1}{2} \sum_{s \in [k]} \hat{g}^k_s$

Claim 6. $\widehat{ALG}^k \geq k/2 + \sum_{s \in [k]} \hat{h}^k_s$

Claim 7. $\hat{\gamma}^k_{i,k} \leq \frac{1}{k} \sum_{j \leq i} \hat{g}^k_j + \hat{h}^k_i$

Claim 8. $k(1 - \hat{x}^k_i) \leq \sum_{j \leq i} \hat{x}^k_j + \sum_{j \leq i} \hat{\gamma}^k_{j,i}$

Claim 9. $1 - \hat{x}^k_i \geq \hat{\gamma}^k_{j,i}$

Constraints (22),(23),(24), (25) and (26) follow trivially. Thus $\hat{s}^k$ is a feasible solution for $LP(k)$. Also observe that the objective value for $LP(k)$ corresponding to the solution $\hat{s}^k$ is $\widehat{ALG}^k/k = ALG^n/qk = ALG^n/n =$ LP-OPT(n). Since $\hat{s}^k$ is a feasible solution to $LP(k)$, LP-OPT$(k) \leq$ LP-OPT(n). □

Lemma 29 above together with Table 2 suffices to prove the following theorem.

Theorem 30. SHUFFLE *attains a factor of at least* 0.560 *in expectation.*

3.3.3 Tightness of the Analysis

In this section we study the tightness of our analysis through a reduction to the analysis of the 2-sided RANKING algorithm presented in Chapter 2 in the context of online bipartite matching in the *Random Order Arrival* (ROA) model . Recall that in this problem we are given one side (say L), of a bipartite graph $G(L \cup R, E)$ while the vertices in the other side arrive online in random order, one vertex at a time. For each incoming vertex its neighborhood in L is told to us and it needs to be matched irrevocably upon arrival. In Chapter 2 we analyzed the RANKING algorithm where we begin by shuffling the vertices in L according to a random permutation thereby assigning a rank to every vertex. We then match each arriving vertex to the highest ranked unmatched vertex in L. We showed the following result.

Theorem 31. *There exists a family of graphs for which* RANKING *attains a factor of* 0.727 *in the ROA model.*

Alternately the above result may be viewed as the analysis of the following randomized algorithm for finding a matching in a given bipartite graph - Randomly permute both bipartitions according to uniform random permutations to assign a distinct rank to each vertex. Iterate through the vertices in L(or R) in increasing order of rank and for each vertex match it to the highest ranked unmatched neighbor in R(or L). We will refer to this algorithm as the 2-sided RANKING algorithm. Lemma 32 given below relates the performance of RANKING and SHUFFLE.

Lemma 32. *For an bipartite graph the expected size of the matching returned by* SHUFFLE *is equal to the expected size of the matching produced by the 2-sided* RANKING *algorithm.*

Proof. Let $G(L \cup R, E)$ be the given bipartite graph. Let $\Upsilon(\Psi)$ be the set of all permutations of $L(R)$. An execution of the 2-sided RANKING algorithm can be characterized by the permutations $\upsilon \in \Upsilon, \psi \in \Psi$ used in the algorithms. Let us denote

this by $Ranking(\upsilon, \psi)$. Similarly we can define Ω to be the set of all permutations of $L \cup R$ and any given execution of SHUFFLE is characterized by a permutation say $\omega \in \Omega$ will be denoted by denoted by $Shuffle(\omega)$. We will prove this result by exhibiting a uniform many-to-one map from ω to $\Upsilon \times \Psi$.

Consider $\omega \in \Omega$. If we look at the relative order of vertices of L in ω, it induces a permutation say $\upsilon \in \Upsilon$. Similarly we can define a permutation $\psi \in \Psi$ on the vertices in R for any given permutation ω. Next we will show that the matching produced by $Ranking(\upsilon, \psi)$ is identical to the matching produced by $Shuffle(\omega)$.

Claim 10. *The matching produced by $Ranking(\upsilon, \psi)$ is identical to the matching produced by $Shuffle(\omega)$.*

Proof. Let x be the first vertex in ω. Without loss of generality let us assume $x \in L$. Thus x is the first vertex in υ. Let y be the highest ranked neighbor for x in ω. By the definition of ψ, y is also the highest ranked neighbor for x in ψ. So both $Shuffle(\omega)$ and $Ranking(\upsilon, \psi)$ will match x to y. The rest of the proof follows easily by induction on the size of the remaining graph. □

Observe that each $\omega \in \Omega$ would be mapped to by the same number of elements in $\Upsilon \times \Psi$. Thus the expected size of the matching returned by SHUFFLE is equal to the expected size of the matching produced by the 2-sided RANKING algorithm. □

Theorem 33 follows from Theorem 31 and Lemma 32.

Theorem 33. *There exists a family of bipartite graphs for which* SHUFFLE *attains a factor of* 0.727.

3.4 Upper Bounds

Theorem 33 stated above bounds the tightness of our analysis of the SHUFFLE algorithm. In this section we will give an upper bound on the performance of a broad class of randomized algorithms called *vertex-iterative* algorithms. This class includes

the SHUFFLE algorithm as well as the algorithm in [3]. We will also prove a slightly weaker bound on the performance of any randomized algorithm as well as for

We call a randomized algorithm for the query commit problem to be *vertex-iterative*(VI) if it iterates through the vertices one at a time in a possibly adaptive sequence and for every vertex scans the edges incident on it in an arbitrary(adaptive) order until it gets matched. A vertex may also choose to scan only a subset of edges incident on it before it decides to "give-up", i.e., after which this vertex will play no further role in the algorithm.

3.4.1 Upper Bound on the Performance of Vertex Iterative Algorithms

In this section we will show a 0.75-upper bound on the performance of any VI algorithm. Our result uses Yao's Lemma [69].

Lemma 34. *[Yao's Lemma] The expected worse-case performance of the optimal randomized algorithm for the query commit problem is upper bounded by the expected performance for the optimal deterministic algorithm for any given distribution over input graphs.*

Thus, we have to come up with a distribution over input graphs and bound the performance of the optimal deterministic VI algorithm for this distribution. We will consider the following family $\mathcal{I}$ of n vertex graphs - For any instance $I \in \mathcal{I}$, the vertex set can be divided into two equal parts C(Clique) and P(Pendant). The vertices in C form a clique while every vertex in P is adjacent to a unique vertex in C. For any vertex $u \in C$, let $u^* \in P$ be the unique pendant vertex adjacent to it. An example is shown in Figure 4. We will assume uniform distribution over graphs in $\mathcal{I}$. Alternately, our distribution may be viewed as the uniform distribution over all relabellings ($\phi : [n] \to [n]$) of vertices of the graph Γ shown in Figure 4.

Unfortunately, it is quite difficult to characterize the optimal deterministic algorithm over graphs in $\mathcal{I}$. Instead we will devise a class of hypothetical algorithms, called

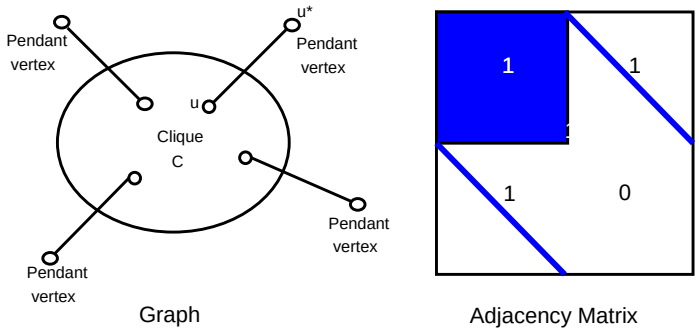

Figure 4: Graph $\Gamma = (C \cup P, E) \in \mathcal{I}$

revealing algorithms whose performance is no-worse than that of any deterministic VI algorithm and then bound their factor. This class of algorithms is hypothetical in the sense that it assumes limited access to the underlying graph $I = (C \cup P, E)$. It uses a notion of *active* and *inactive* vertices and at the start of the algorithm all vertices are marked to be active.

A revealing algorithm is a VI algorithm that proceeds in several phases. We begin every phase by choosing an arbitrary active vertex say v. If $v \in P$ then it gets matched to its unique neighbor in $v^* \in C$ and we terminate the phase by marking both v and v^* as being inactive. Otherwise if $v \in C$ then it starts querying edges incident to other active vertices (according to a possibly adaptive strategy). If we query an edge incident to a vertex $u^* \in P$ then the u^* *reveals* its unique neighbor $u \in C$ and gets matched to it and both u and u^* are marked as inactive. However if v queries another active vertex $w \in C$, then v and w get matched, as required in the query commit model, and $\{v, w, v^*, w^*\}$ are marked to be inactive. v may also choose to give-up after trying a few edges, at which point it is marked as inactive (we also mark $v^* \in P$ as inactive at this point). The phase ends when v gets marked as inactive and it is not considered any further by the algorithm. The algorithm terminates when all vertices are marked to be inactive.

Let $\mathcal{O}$ be the optimal deterministic VI algorithm for instances drawn from $\mathcal{I}$. In Lemma 35, we show that there exists a deterministic revealing-algorithm does at least

as well as $\mathcal{O}$.

Lemma 35. *There exists a deterministic revealing-algorithm that does at least as well as $\mathcal{O}$ on every instance $I \in \mathcal{I}$.*

Proof. Consider the execution of $\mathcal{O}$ on any instance $I = (C \cup P, E)$. We can simulate the execution of $\mathcal{O}$ on I by a revealing algorithm $\mathcal{A}$. The only possible difficulty could be if $\mathcal{O}$ scans an edge incident on a vertex that is already marked to be inactive by $\mathcal{A}$. This step is not allowed in $\mathcal{A}$, and cannot be simulated. In this case the current vertex in our simulation immediately gives up and we proceed to the next phase. One can check that under this simulation $\mathcal{A}$ produces a matching that is at least as large as the one returned by $\mathcal{O}$. □

Corollary 36 follows from Lemma 35.

Corollary 36. *For an instance drawn uniformly at random from $\mathcal{I}$ the expected performance of the optimal deterministic revealing-algorithm is at least as good as the expected performance of the optimal deterministic* VI *algorithm.*

Let $\mathcal{A}$ be the optimal deterministic revealing-algorithm for graphs drawn from $\mathcal{I}$. By Lemma 34 and Corollary 36 we only need to bound performance of $\mathcal{A}$. To get a handle on this we define a randomized revealing algorithm Random-Reveal(RR) and then in Lemma 37 we show that the expected performance of RR for the graph Γ shown in Figure 4 is (approximately) equal to the expected performance of $\mathcal{A}$ on graphs drawn uniformly at random from $\mathcal{I}$. Thus we can use the analysis of RR to upper bound the performance of any VI algorithm.

RR is defined as follows - For any given instance $I = (C \cup P, E)$ we choose a random permutation ρ of $C \cup P$. Then we process the vertices one at a time according to ρ. If the current vertex is inactive then it is ignored else we query edges incident on it in random order until it gets matched. The algorithm terminates when there are fewer than $n^{0.9}$ active vertices.

Remark 2. *The modified termination condition changes the performance of the algorithm by only a negligible factor and is used to simplify the proof.*

Lemma 37. *For the graph* Γ *shown in Figure 4*

$$\mathop{\mathbb{E}}_{\rho}[RR(\Gamma)] = \mathop{\mathbb{E}}_{I \in \mathcal{I}}[\mathcal{A}(I)]$$

Proof. Without loss of generality we may assume that $\mathcal{A}$ is a *greedy* algorithm in the sense that it will never cause a vertex to give-up until it gets matched or has exhausted all its options. The lemma can be proved by establishing the following fact which follows by induction and using the observation that at the end of any phase, the induced graph on the set of active vertices is uniformly distributed over graphs in $\mathcal{I}$ but defined over smaller number of vertices.

Fact: The distribution of the induced graphs over unmatched vertices at the end of the k^{th} phase is identical for both RR and $\mathcal{A}$ for every choice of $k \in [n]$. □

Now we are left to analyze the performance of RR on Γ. We do this in the subsequent lemma.

Lemma 38. *For the graph* Γ *shown in Figure 4*

$$\mathop{\mathbb{E}}_{\rho}[RR(\Gamma)] = 0.75$$

Proof. Note that Γ has a unique optimal solution where all n vertices get matched, namely $\{ww^* \mid \forall w \in C\}$. We will argue that for an arbitrary phase, the expected number of vertices that get matched is $3/4$ of the expected number of vertices that are labeled as inactive during the phase. This will establish the above lemma.

For a given phase let v be the active vertex chosen by RR at the start of this phase. The proof for this case is based on the following three simple claims.

Claim 11. $Pr[v \in C] = Pr[v \in P]$

Proof. At any stage of a revealing algorithm the number of active vertices belonging to C is equal to the number of active vertices in P. The claim follows since v is chosen at random from all the active vertices. □

Claim 12. *If $v \in P$, then two vertices get matched by* RR *in the current phase and exactly these two vertices are also labeled to be inactive.*

Proof. Follows from the definition of a revealing-algorithm. □

Claim 13. *If $v \in C$, then* 4 *vertices get matched in expectation and on an average we label* 6 *more vertices as being inactive in this phase.*

Proof. First observe that since we truncate the execution of the algorithm to the point when there are at least $n^{0.9}$ active vertices, v will match v^* with negligible probability. We will ignore the probability of this event for the rest of the proof, since this can only introduce a negligible error.

Recall that in the RR algorithm the given vertex v will randomly probe all its active neighbors until it gets matched. If it probes a neighbor belongs to P the neighbor would reveal itself and get matched to its partner in the optimal solution. Thus v will continue to probe its neighbors until it probes a vertex in C. Since in any revealing algorithm the number of active vertices belonging to C is equal to the number of active vertices in P the average number of vertices in P that get probed by v is given by $\frac{1}{2}\sum_{i=1} \frac{i}{2^i} = 1$.

Thus on an average v will cause 1 vertex in P to be revealed before it gets matched. This vertex will go on to match its partner in the optimal solution. So in expectation we will match 4 vertices (2 edges) in any given phase. By similar arguments it follows that we will mark at most 6 vertices to be inactive in expectation.This is shown in Figure 5. □

By Claims 11, 12, 13 we see that the the expected number of vertices that get matched in any phase is given by $2(0.5) + 4(0.5) = 3$ while the expected number of

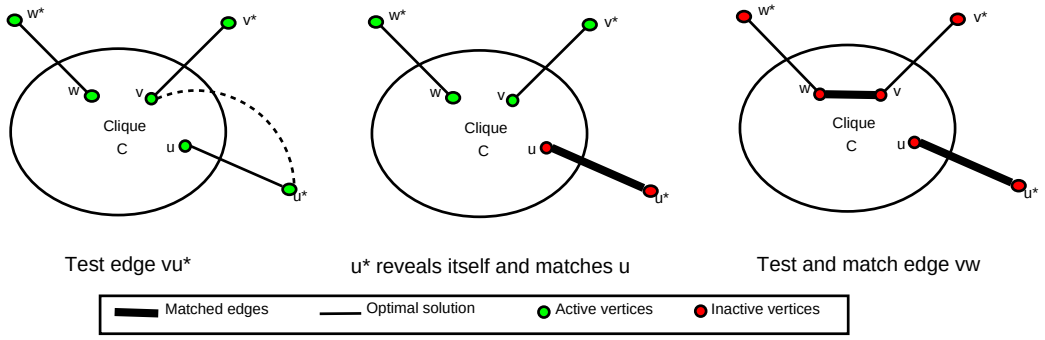

Figure 5: Expected behavior of RR when $v \in C$

vertices that are marked as inactive is $2(0.5) + 6(0.5) = 4$. Thus RR attains a factor of $3/4$. □

From Lemmas 35 and 37 we have the following theorem.

Theorem 39. *There exists a family of graphs for which no* VI *algorithm attains a factor better than* 0.75*.*

3.4.2 Upper Bound on the Performance of any Randomized Algorithm

In this section we establish an upper bound on the performance of any randomized algorithm. As before our analysis relies on Yao's lemma. The bound presented is slightly weaker than that presented for VI algorithms in Theorem 39, though it applies to any randomized algorithm.

Theorem 40. *No randomized algorithm can attain a factor better than* $19/24 = 0.7916$ *for the query commit problem.*

Proof. Consider a graph H over 4 vertices a, b, c, d with edge set $\{ab, bc, ca, da\}$, i.e. it looks like a triangle with one pendant vertex. Consider the distribution of graphs produced by uniformly permuting the labels on the vertices. This results in a distribution over $4! = 24$ isomorphic graphs. Now let us characterize the optimal deterministic algorithm for this distribution.

Since in the beginning, the algorithm knows nothing about the graph, without loss of generality we can assume it queries an arbitrary edge. Two cases may arise, either the edge is present or not. The first case happens with probability $4/6 = 2/3$, following which the algorithm can only probe the remaining pair of vertices and check if there is an edge between them. It is not difficult to see that given that the first edge that is scanned is found to be present, we find the optimal matching (of size 2) with probability 1/2 and return a suboptimal solution (of size 1) with probability 1/2.

On the other hand, if the first edge that is scanned, say $\alpha\beta$, is not present then it will imply that there is an edge connecting the other pair of vertices (say γ, δ). Clearly the optimal deterministic algorithm won't scan $\gamma\delta$ as that will give matching of size 1 only. Thus in the next step, the optimal algorithm will query an edge with one end-point in $\{\alpha, \beta\}$ and the other end-point in $\{\gamma, \delta\}$.

Observe that if the second query is also unsuccessful (this happens with probability 1/4), we can identify the pendant vertex and find the optimal solution. On the other hand if the second query results in a match then the next query can only check for an edge between the remaining two vertices. In this case, it is not difficult to see that we get an optimal solution with probability 2/3 and a suboptimal solution with probability 1/3. Thus the expected size of the solution returned by the optimal deterministic algorithm is $2/3\,[2(1/2) + 1(1/2)] + 1/3\,[2(1/4) + 0.75(2(2/3) + 1(1/3))] = 19/12$. Since the size of the optimal solution is 2, the optimal deterministic algorithm for graphs drawn from the aforementioned distribution attains a factor of 19/24. The result follows by applying Yao's lemma. □

3.5 Omitted Proofs

Proof of Claim 2

Proof. Since u is unmatched in $Shuffle(\rho)$ and it is adjacent to w(by our assumption), w must be matched in $Shuffle(\rho)$. Let v be the vertex that w gets matched to. Since we are in Case 2 of Lemma 26, $v \neq w^*$. Next let us look at $Shuffle(\rho')$; v must available when w gets matched to u. Thus when w gets matched in $Shuffle(\rho)$ it has at least two options i.e. u and v.

Finally lets consider $\rho'' = \rho'[w^* \rightsquigarrow s]$. By the stability property (Claim 1) the matchings produced by $Shuffle(\rho')$ and $Shuffle(\rho'')$ at the time when w is being considered differ by an augmenting path. Thus the number of options available to w in $Shuffle(\rho')$ and $Shuffle(\rho'')$ can differ by at most 1. Recall that both u and $v \neq w^*$ were available to w in $Shuffle(\rho')$. So at least one of them should still be available in $Shuffle(\rho'')$. Therefore w will get matched in $Shuffle(\rho'')$. □

Proof of Claim 3

Proof. Let us consider two cases.

Case 1: $\rho'^{-1}(w) > t$. Since u gets matched to w in $Shuffle(\rho')$, w should be unmatched until time t. Clearly the execution of $Shuffle(\rho'[w^* \rightsquigarrow s])$ and $Shuffle(\rho')$ would be identical until time s. Thus when we consider w^* in $Shuffle(\rho'[w^* \rightsquigarrow s])$, w is still unmatched. Therefore w^* will surely get matched by $Shuffle(\rho'[w^* \rightsquigarrow s])$.

Case 2: $\rho'^{-1}(w) \leq t$. If possible let w^* not get matched in $Shuffle(\rho'[w^* \rightsquigarrow s])$. Let $r = \rho'^{-1}(w)$. In this case $Shuffle(\rho')$ and $Shuffle(\rho'[w^* \rightsquigarrow s])$ are identical until time r. Therefore both u and w^* must be unmatched when we consider w in $Shuffle(\rho'[w^* \rightsquigarrow s])$. Since $s \leq t$, w would choose w^* instead of u which is a contradiction. □

Proof of Claim 4

Proof.

$$\begin{aligned}
\widehat{ALG}^k &= \frac{ALG^n}{q} && \text{Using equation (30)} \\
&= \sum_{s\in[n]} x_s^n/q && \text{Using equation (16)} \\
&= \sum_{i\in[k]} \sum_{s:\lfloor s/q\rfloor=i} x_s^n/q \\
&= \sum_{i\in[k]} \hat{x}_i^k && \text{Using equation (27)}
\end{aligned}$$

□

Proof of Claim 5

Proof.

$$\begin{aligned}
\widehat{ALG}^k &= \frac{ALG^n}{q} && \text{Using equation (30)} \\
&= \frac{\frac{n}{2} + \frac{1}{2}\sum_{s\in[n]} g_s^n}{q} && \text{Using equation (17)} \\
&= \frac{k}{2} + \frac{1}{2q}\sum_{t\in[n]} g_t^n \\
&= \frac{k}{2} + \frac{1}{2q}\sum_{i\in[k]} \sum_{s:\lfloor s/q\rfloor=i} g_s^n \\
&= \frac{k}{2} + \frac{1}{2}\sum_{i\in[k]} \hat{g}_i^k && \text{Using equation (28)}
\end{aligned}$$

□

Proof of Claim 6

Proof.

$$\begin{aligned}
\widehat{ALG}^k &= \frac{ALG^n}{q} && \text{Using equation (30)} \\
&= \frac{\frac{n}{2} + \sum_{s\in[n]} h_s^n}{q} && \text{Using equation (18)}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{k}{2} + \frac{1}{q}\sum_{t\in[n]} h_t^n \\
&= \frac{k}{2} + \frac{1}{q}\sum_{i\in[k]}\sum_{s:\lfloor s/q\rfloor=i} h_s^n \\
&= \frac{k}{2} + \sum_{i\in[k]} \hat{h}_i^k && \text{Using equation (28)}
\end{aligned}
$$

□

Proof of Claim 7

Proof.

$$
\begin{aligned}
\hat{\gamma}_{i,k}^k &= \frac{\sum_{s:\lfloor s/q\rfloor=i}\sum_{t:\lfloor t/q\rfloor=n} \gamma_{i,j}^n}{q^2} && \text{Using equation (29)} \\
&\leq \frac{\sum_{s:\lfloor s/q\rfloor=i} \gamma_{s,n}^n}{q} && \text{Using equation (23)} \\
&\leq \frac{\sum_{s:\lfloor s/q\rfloor=i} \frac{1}{n}\sum_{t\leq s} g_t^n + h_s^n}{q} && \text{Using equation (19)} \\
&= \frac{\sum_{s:\lfloor s/q\rfloor=i}\sum_{t\leq s} g_t^n}{qn} + \hat{h}_i^k \\
&\leq \frac{\sum_{s:\lfloor s/q\rfloor=i} g_s^n}{n} + \hat{h}_i^k \\
&\leq \frac{q\sum_{j\leq i} \hat{g}_j^k}{n} + \hat{h}_i^k \\
&\leq \frac{1}{k}\sum_{j\leq i} \hat{g}_j^k + \hat{h}_i^k
\end{aligned}
$$

□

Proof of Claim 8

Proof.

$$
k(1-\hat{x}_j^k) = \frac{k\sum_{t:\lfloor t/q\rfloor=i}(1-x_t^n)}{q} \qquad \text{Using equation (27)}
$$

$$
\begin{aligned}
&\leq \frac{n \sum_{t:\lfloor t/q \rfloor = i} \left(\sum_{s \leq t} x_s^n + \sum_{s \leq t} \gamma_{s,t}^n \right)}{q^2} && \text{Using equation (20)} \\
&\leq \frac{\sum_{s \leq iq} x_s^n}{q} + \frac{\sum_{t:\lfloor t/q \rfloor = i} \sum_{s \leq t} \gamma_{s,t}^n}{q^2} \\
&= \sum_{j \leq i} \hat{x}_j^k + \sum_{j \leq i} \hat{\gamma}_{j,i}^k && \text{Using equations (27) and (29)}
\end{aligned}
$$

□

Proof of Claim 9

Proof.

$$
\begin{aligned}
1 - \hat{x}_i^k &= \frac{\sum_{t:\lfloor t/q \rfloor = i} 1 - x_t^n}{q} && \text{Using equation (27)} \\
&\geq \frac{\sum_{t:\lfloor t/q \rfloor = i} \sum_{s:\lfloor s/q \rfloor = j} \gamma_{s,t}^n}{q^2} && \text{Using equation (21)} \\
&\geq \hat{\gamma}_{j,i}^k
\end{aligned}
$$

□

3.6 Conclusion and Open Problems

In this chapter we discussed the oblivious query commit model which attempts to capture the situation where the lack of information constraints us to implement myopic, greedy-like algorithms. A simple deterministic greedy algorithm achieves a factor 1/2 which is tight for deterministic algorithms. We took the first steps towards answering the big open question in this area that whether a randomized greedy algorithm has a significantly better approximation factor. We analyzed the SHUFFLE algorithm which is a generalization of the RANKING algorithm described in Chapter 2, and proved that it attains a factor of at least 0.56. We also presented upper bounds on performance of a broad class of randomized algorithms.

The main open question left in this line work is whether SHUFFLE is really the optimal randomized algorithm for this problem. At a higher level this pertains to the correct way of infusing randomness in to the greedy algorithm. Also, it is strange that the worst examples for SHUFFLE that we could find are bipartite graphs (refer Section 3.3.3), since bipartite graphs actually satisfy the monotonicity property mentioned in Section 3.2.2. This leaves the tantalizing possibility that SHUFFLE performs as well on general graphs as RANKING does on bipartite graphs, and therefore achieves a much greater than 0.56. Resolving this is an interesting technical challenge.

CHAPTER IV

MATCHING IN STOCHASTIC QUERY COMMIT MODEL

4.1 Introduction

In this chapter we will discuss a stochastic variant of the Oblivious Query Commit Model presented in Chapter 3. This is motivated by barter exchange problems where we have additional stochastic information about the given data. For example in the Kidney exchange problem mentioned earlier, based on cheap blood tests, it is possible to estimate the likelihood of two patients being compatible without explicitly testing for compatibility. We refer the reader to [54, 55] for more details.

Concretely we will study the following problem - For p, a probability vector indexed by pairs of vertices from a vertex set V, let $G(V, p)$ denote an undirected Erdős-Rényi graph on V. That is, for any $(u, v) \in V \times V$, $p_{uv} = p_{vu}$ denotes the (known) probability that there is an edge connecting u and v in G. For every pair $(u, v) \in V \times V$ we are *not* told a priori whether there is an edge connecting these vertices, until we *probe/scan* this pair. If we scan a pair of vertices and find that there is an edge connecting them we are constrained to *pick* this edge and in this case both u and v are removed from the graph. However, if we find that u and v are not connected by an edge, they continue to be available (to others) in the future. The goal is to maximize the number of vertices that get matched.

We will refer to the above as the Stochastic Query Commit Problem (SQCP), since whenever we probe a pair of adjacent vertices, we are committed to picking them. Once again the performance of our algorithm is compared against the optimal offline algorithm that knows the underlying graph for each instantiation of the problem and finds the maximum matching in it. Note that since the input is itself random,

the average performance of the optimal offline-algorithm is the expected size of the maximum matching in this random graph.

4.1.1 Our Results

It is easy to see that the simple greedy algorithm, which probes pairs in an arbitrary order, would return a *maximal* matching in every instance of the problem and is therefore a factor 0.5 approximation algorithm. We give a sampling based algorithm for this problem that does better than this and prove the following results.

(a) There exists a randomized algorithm that attains a competitive ratio of at least 0.575 for the Stochastic Matching with Commitment Problem that runs in time $\tilde{O}(n^4)$ for a graph with n vertices. Furthermore, the running time can be reduced to $\tilde{O}(n^3)$ in the case where the expected size of the optimal matching is a positive fraction of the number of vertices in the graph.

(b) No algorithm can attain a competitive ratio better than 0.896 for the SQCP.

Our algorithm uses *offline simulations* to determine the relative importance of edges to decide the order in which to scan them. It is based on a novel sampling lemma that might be of independent interest in tackling online optimization settings, wherein an algorithm needs to make irrevocable online decisions with limited stochastic knowledge. This sampling trick is explained in section 4.2.3. Even though the proof for our sampling lemma is based on solving an exponentially large linear program, we also give a fast combinatorial algorithm for it. As a result our algorithm can be implemented in time $O(n^3)$.

4.1.2 Related Models and Results

The problem has similar flavor to several well known stochastic optimization problems such as the stochastic knapsack [13] and the shortest-path [50, 52]; refer to [61] for a detailed discussion on these problems. As stated earlier SQCP is a relaxation of

the oblivious query commit problem discussed in Chapter 3. Thus the lower bound of 0.56 presented in Section 3.3 carries over to SQCP.

The SQCP has also been studied in another more general model first introduced by Chen et al. [11]. In their model every vertex $v \in V$ had a *patience parameter* $t(v)$ indicating the maximum number of failed probes v is willing to participate in. After $t(v)$ failed attempts, vertex v would leave the system, and would not be considered for any further matches. Our model can be viewed as a special case of their setting where $t(v) = n$ for every vertex. However Chen et. al., and subsequently Bansal et al. [5], compared their performance against the optimal online algorithm. This was necessary because if we consider the case of the star graph, where each edge has a probability of $1/n$ and $t(v) = 1$ for every vertex v, then any online algorithm can match the center of the star with probability at most $1/n$, while there exists an edge incident on the center with probability $1 - 1/e$. In contrast, our results are against the strongest adversary, i.e., the optimal offline omniscient algorithm. Clearly the performance of the optimal online algorithm can be no better than that of such an omniscient algorithm.

In their model [11], Chen et al. presented a $1/4$ competitive algorithm. Their results were later extended to the weighted setting by Bansal et al. [5] who used a linear program to bound the performance of the optimal algorithm and gave a $1/4$ competitive algorithm for the general case, and a $1/3$ competitive algorithm for the special case of bipartite graphs. It is interesting to note that the linear program considered by Bansal et al. has an integrality gap of 2 for general graphs which limits the best factor attainable by LP based algorithms. Another interesting aspect of their model is that the optimal online algorithm is a stochastic dynamic program having exponentially many states, and it is not even known if the problem is NP-hard when the patience parameters $t(.)$, can be arbitrary.

4.1.3 Technical Contributions

Observe that the simple algorithm, which weighs (or probes) an edge e according to probability p_e, is not necessarily the best way to proceed. Consider a path having 3 edges such that the middle edge is present with probability 1 whereas the other two edges are each present with probability 0.9. Even though the middle edge is always present, it is unlikely to be involved in any maximum matching. Conversely, the outer edges will always be a part of some maximum matching when they appear.

In order to quantify the relative importance of an edge e, we define the quantity q_e as the probability that the edge belongs to a maximum matching. This can be calculated by sampling a collection of representative graphs from the given distribution; we provide further details in Section 4.4.1 on how to implement this efficiently. Note that this is done as a preprocessing step *without* probing any of the edges in the given graph (a necessary requirement, as probing an edge could lead to unwanted commitments). Clearly the probability that a vertex would get matched in the optimal solution is the sum of q_e^* for all edges incident on it and this gives us a way to approximate the optimal solution.

Similarly we can also calculate the conditional probability that an edge belongs to the maximum matching, given that it is present in the underlying graph. We use this as a measure of the importance of the edge. Observe that it is safe to probe edges where this conditional probability is large, since we are unlikely to make a mistake on such edges. After we are done probing these edges we are left with a residual graph where this conditional probability is small for every edge.

Ideally at this point what we would like to do is to simulate the fractional matching given by the q_e^*, i.e., include every edge with probability q_e^*. However, this is made impossible by the combination of our lack of knowledge of the graph and the commitments we are forced to make as we scan edges to obtain information about the graph. To overcome these limitations, we devise a novel sampling technique, described in

section 4.2.3, that gives us a partial simulation. This sampling algorithm outputs a (randomized) ordering to scan the edges incident to a given vertex, so as to ensure that edge e is included with probability at least some large positive fraction of q_e^*.

4.2 Preliminaries

4.2.1 Problem Statement

We are given a set of vertices V, and for every unordered pair of vertices $u, v \in V$, we have a (known) probability p_{uv} of the edge (u, v) being present. These probabilities are independent over the edges. Let $\mathcal{D}$ denote this distribution over all graphs defined by p. Let $G(V, E)$ be a graph drawn from $\mathcal{D}$. We are given only the vertex set V of G, but the edge-set E is not revealed to us unless we *scan* an unordered pair of vertices. A pair $(u, v) \in V \times V$ may be scanned to check if they are adjacent and if so then they are matched and removed from the graph. The objective is to maximize the expected number of vertices that get matched.

We compare our performance to the optimal off-line algorithm that knows the edges before hand, and reports the maximum matching in the underlying graph. We say an online algorithm $\mathcal{A}$ attains a competitive ratio of γ for the SQCP if, for every problem instance $\mathcal{I} = (G(V, .), p)$, the expected size of the matching returned by $\mathcal{A}$ is at least γ times the expected size of the optimal matching in the Erdős Rényi graph $G(V, p)$. That is, $\gamma = \min_{\mathcal{I}=(G(V,.),p)} \left\{ \frac{E[\mathcal{A}(\mathcal{I})]}{E[\text{max matching in } G(V,p)]} \right\}$.

4.2.2 Definitions

For any graph H drawn from $\mathcal{D}$, let $M(H)$ be an arbitrarily chosen maximum matching on H. We define

$$q_{uv}^* = \Pr_{H \leftarrow \mathcal{D}} (u \sim v \text{ in } M(H)).$$

Clearly $q_{uv}^* \leq p_{uv}$, since an edge cannot be part of a maximal matching unless it is actually in the graph. In general, the ratio q_{uv}^*/p_{uv} can be thought of as the

conditional probability that an edge is in the matching, given that it appears in the graph. For a given vertex u, the probability that u is matched in M is exactly

$$Q_u(G) := \sum_v q^*_{uv}.$$

This of course is at most 1. We will compare the performance of our algorithm against the expected size of a maximum matching (denoted by OPT) for a graph drawn from $\mathcal{D}$. Thus we have,

$$\mathbb{E}[|\text{OPT}|] = \mathbb{E}\left[|M(H)|\right] = \frac{1}{2}\sum_u Q_u(G) = \sum_{(u,v)} q^*_{uv}, \tag{31}$$

where the last sum is taken over unordered pairs. Finally define an unordered pair (u, v) to be a *candidate edge* if both u and v are still unmatched and (u, v) is yet to be scanned. At any stage let $F(G) \subseteq V \times V$ be the set of candidate edges, and for any $u \in V$, let $N(u, G) \subseteq F(G)$ denote the candidate edges incident on u. A vertex u is defined to be *alive* if $|N(u, G)| > 0$.

4.2.3 Sampling Technique

In this section we will describe a sampling technique that will be an important component of our algorithm. A curious reader may directly read Corollary 43 and proceed to Section 4.3 to see an application of this technique. Frequently over the course of our algorithm we will encounter the following framework: We have a vertex u, whose incident edges have known probabilities p_{uv} of being connected to u. We would like to choose an ordering on the incident edges to probe accordingly so that each edge is included(scanned and found to be present) with some target probability of at least r_{uv} (which may depend on v).

Clearly there are some restrictions on the r_{uv} in order for this to be feasible; for example the situation is clearly hopeless if $r_{uv} > p_{uv}$. More generally, for each subset S of the neighborhood of u, it must be the case that the sum of the target probabilities of vertices in S (the desired probability of choosing some member of S) is at most

the probability that at least one vertex of S is adjacent to u. As it turns out, these are the *only* necessary restrictions.

Lemma 41. *Let $A_1, A_2, \ldots A_k$ be independent events having probabilities $p_1, \ldots, p_k$. Let $r_1, \ldots, r_k$ be fixed non-negative constants such that for every $S \subseteq \{1, \ldots, k\}$ we have*

$$\sum_{i \in S} r_i \leq 1 - \prod_{i \in S}(1 - p_i). \tag{32}$$

Then there is a probability distribution over permutations π of $\{1, 2, \ldots, k\}$ such that for each i, we have

$$\mathrm{P}(A_i \text{ is the earliest occurring event in } \pi) \geq r_i\,. \tag{33}$$

Proof. By the Theorem of the Alternative from Linear Duality [18], it suffices to show that the following system of $n! + 1$ inequalities in $n + 1$ variables $\{x_1, \ldots, x_n, y\}$ does not have a non-negative solution:

$$y - \sum_k x_k r_k \;<\; 0 \tag{34}$$

$$\forall \pi \in S_n, \qquad y - \sum_k x_k p_k \prod_{\substack{j<k \\ \text{in}\pi}}(1 - p_j) \;\geq\; 0 \tag{35}$$

Assume such a solution exists. Without loss of generality we may assume $x_1 \geq x_2 \geq \cdots \geq x_n \geq 0$. Combining the first inequality with the inequality from the identity permutation, we have

$$\sum_{i=1}^{n} x_i p_i \prod_{j=1}^{i-1}(1 - p_j) < \sum_{k=1}^{n} x_k r_k. \tag{36}$$

On the other hand, we have for each k by applying equation (32) to $S = \{1, 2, \ldots k\}$ that

$$\sum_{i=1}^{k} r_i \leq 1 - \prod_{j=1}^{k}(1 - p_j).$$

By weighting each of these equations by $(x_i - x_{i+1})$ and treating $x_{n+1} = 0$ (note that each of these weights are nonnegative by assumption) and adding, we obtain

$$\sum_{k=1}^{n} x_k r_k \leq \sum_{k=1}^{n}(x_k - x_{k+1})[1 - \prod_{j=1}^{k}(1 - p_j)]. \tag{37}$$

It can be checked directly that both the left side of equation (36) and the right hand side of equation (37) are equal to

$$\sum_{\substack{S\subseteq\{1,2,\ldots n\}\\ S\neq\emptyset}} (-1)^{|S|-1} x_{\max(S)} \prod_{i\in S} p_i,$$

implying that the two equations contradict each other. Therefore no such solution to the dual system can exist, so the original system must have been feasible. □

In theory, it is possible to find the desired distribution π using linear programming. However, it turns out there is a faster constructive combinatorial algorithm:

Lemma 42. *A probability distribution π on permutations solving the program (33) can be constructively found in time $O(n^2)$.*

We defer the proof of this lemma and a description of the relevant algorithm to Section 4.6. The following corollary follows immediately from Lemma 42.

Corollary 43. *Given a graph $G(V, E)$ and $u \in V$, such that $q^*_e/p_e < \alpha < 1$ for every $e \in N(u, G)$, there exists a randomized algorithm for scanning the edges in $N(u, G)$ in such a way that the probability of including any edge $e \in N(u, G)$ in the matching is at least $\delta(u, G)q^*_e$, where*

$$\delta(u, G) = \frac{1 - \exp(-\sum_{v\in N(u,G)} q^*_{uv}/\alpha)}{\sum_{v\in N(u,G)} q^*_{uv}}.$$

Proof. Note that for any $u \in V$, and $S \subseteq N(u, G)$, $1 - \prod_{v\in S}(1 - p_{uv}) \geq \sum_{v\in S} q^*_{uv}$, since the right side represents the probability q is matched to S in our chosen maximal matching and the left side the probability that there is at least one edge connecting q to S. Thus (p, q^*) satisfy the condition for Lemma 41. However, we can do better. For any given S, if we scale each q_e by $\left(1 - \prod_{v\in S}(1 - p_{uv})\right) / \sum_{v\in S} q^*_{uv}$, the above condition still remains satisfied for that S. Since $q^*_e/p_e < \alpha$ we have

$$\frac{1 - \prod_{v\in S}(1 - p_{uv})}{\sum_{v\in S} q^*_{uv}} \geq \frac{1 - \exp(-\sum_{v\in S} p_{uv})}{\sum_{v\in S} q^*_{uv}} \geq \frac{1 - \exp(-\sum_{v\in S} q^*_{uv}/\alpha)}{\sum_{v\in S} q^*_{uv}} \tag{38a}$$

$$\geq \frac{1-\exp(-\sum_{v\in N(u,G)} q^*_{uv}/\alpha)}{\sum_{v\in N(u,G)} q^*_{uv}} = \delta(u,G)\,, \tag{38b}$$

and equation (38b) follows since $1-\exp(-\sum_{v\in S} q^*_{uv}/\alpha)/\sum_{v\in S} q^*_{uv}$ is a decreasing function in $\sum_{v\in S} q^*_{uv}$, thus achieving its minimum value at $S = N(u,G)$. Therefore we can replace our q^* by $\delta(u,G)q^*$ and still have the conditions of Lemma 41 hold. □

4.3 Matching Algorithm on Unweighted Erdős-Rényi graphs

Our algorithm can be divided into two stages. The first stage involves several iterations each consisting of two steps - Estimation and Pruning. The parameters α and C will be determined in Section 4.4.

- **Step 1 (Estimation)**: Generate samples $H_1, H_2, \ldots H_C$ of the Erdős-Rényi graph by sampling from $\mathcal{D}$. For each sample, generate the corresponding maximum matching $M(H_j)$. For every prospective edge (u,v), let q_{uv} be the proportion of samples in which the edge (u,v) is contained in $M(H_j)$.
- **Step 2 (Pruning)**: Let (u,v) be an edge having maximum (finite) ratio q_{uv}/p_{uv}. If this ratio is less than α, end Stage 1. Otherwise, scan (u,v). If (u,v) is present, add it to the partial matching; remove u and v from V, and return to Step 1; otherwise set p_{uv} to 0 and return to Step 1.

We recompute q_{uv} every time we scan an edge. Stage 1 ends when the maximum (finite) value of q_e/p_e falls below α. Note that at this point we stop recomputing q, and these values of q will remain the same for each pair of vertices for the remainder of the algorithm. We now describe the second stage of the algorithm.

The second stage also has several iterations each consisting of two steps. At the start of this stage define $X = V$.

- **Step 1 (Random Bipartition)**: Randomly partition X into two equal sized sets L and R and let B be the *bipartite graph* induced by L and R.

- **Step 2 (Sample and Match)**: Iterate through the vertices in L in an arbitrary order, and for each vertex $u \in L$ sample a neighbor in $N(u, B)$ by choosing a vertex in R using the sampling technique described in Corollary 43[1]. At the end redefine X to be the set of alive vertices in R and discard the unmatched vertices in L. Recall that a vertex was defined to be alive if it is still unmatched and it has at least one candidate edge incident on it.

The algorithm terminates when X becomes empty.

4.4 Analysis of the Sampling based Algorithm

In this section we will analyze the competitive ratio for the algorithm described earlier. We begin by analyzing Stage 1 of the algorithm. For each iteration in Stage 1, define the residual graph at the start of the i^{th} iteration to be G_i starting with $G_1 = G$. We denote by $q^*_{e,i}$ the actual probability that e is contained in the maximal matching on G_i and $q_{e,i}$ as our estimate calculated in Step 1. We define

$$\epsilon_e := \max_i |q_{e,i} - q^*_{e,i}| .$$

Let the total number of iterations in this stage be k and let $G' = G_k$. Let ALG_1 be the set of edges that are matched in Stage 1 and let $OPT(G_i)$ be the optimal solution in the residual graph at the start of the i^{th} iteration. We have the following lemma.

Lemma 44.

$$\mathbb{E}\left[|OPT| - |OPT(G')|\right] \leq (2-\alpha)\mathbb{E}[|ALG_1|] + \sum_e \epsilon_e .$$

Proof. For $i \in [k]$, let $Gain(i)$ be 1 if the edge scanned in the i^{th} iteration is present, and 0 otherwise. We will first show that $E\left[|OPT(G_i)| - |OPT(G_{i+1})|\right] \leq (2-\alpha)\mathbb{E}[Gain(i)]$. Three cases may arise during the i^{th} iteration.

[1]The algorithm described in Corollary 43 requires the exact estimates for q^*_e. However we will show in our analysis that for large enough samples C, q_e defined above is a good estimate of q^*_e.

- Case 1: The edge scanned in the i^{th} iteration is not present. Then $OPT(G_i) = OPT(G_{i+1})$ and $Gain(i) = 0$ thus, $|OPT(G_i)| - |OPT(G_{i+1})| = Gain(i) = 0$.
- Case 2.1: The edge scanned in the i^{th} iteration is present but does not belong to $OPT(G_{i+1})$. This happens with probability $p_e - q^*_{e,i}$. Then $|OPT(G_i)| - |OPT(G_{i+1})| = 2$ and $Gain(i) = 1$.
- Case 2.2: The edge scanned in the i^{th} iteration is present and belongs to $OPT(G_i)$. This happens with probability $q^*_{e,i}$. Then $|OPT(G_i)| - |OPT(G_{i+1})| = 1$ and $Gain(i) = 1$.

Summing over all three cases, we see that

$$\mathbb{E}[|OPT(G_i)| - |OPT(G_{i+1})|] = 2(p_e - q^*_{e,i}) + q^*_{e,i} \leq p_e(2 - \alpha) + \epsilon_e \ ,$$

while the expected gain from scanning the edge is simply p_e. The result follows from adding over all scanned edges, and noting for the additive factor that each edge is scanned at most once in the first stage (and indeed in the whole algorithm). □

Analysis of Stage 2: Let us begin by analyzing the first iteration of the second stage of the algorithm. The analysis for the subsequent iterations would follow along similar lines. Let G' be the residual graph at the start of the second stage, where $q_e/p_e < \alpha$ for every candidate edge e, and $1/2 \sum_u Q_u(G') = \mathbb{E}[|OPT(G')|]$. The following lemma bounds the performance of the first iteration of Stage 2 on G'.

Lemma 45. *The expected number of edges that are matched in the first iteration of Stage 2 of the algorithm is at least* $\left(1 - \frac{1}{e}\right)\left(1 - e^{-1/2\alpha}\right)|OPT(G')| - \sum_e \epsilon_e$.

Proof. Let us define an indicator random variable Y_u for every $u \in V$ that is 1 if $u \in R$ and 0 otherwise and $\Pr[Y_u = 1] = \Pr[Y_u = 0] = 1/2$. Thus $\mathbb{E}[Y_u] = 1/2$. We will use $u \sim v$ to denote that u and v get matched. The expected number of vertices in R that will get matched in the first iteration is given by

$$\mathbb{E}\left[\text{matched vertices in } \ R\right] \tag{39a}$$

$$= \mathbb{E}_Y \left[\sum_u Y_u \left\{ 1 - \prod_{v \neq u} \left(1 - Pr\left[u \sim v,\ v \in L \mid u \in R\right]\right) \right\} \right] \tag{39b}$$

$$= \mathbb{E}_Y \left[\sum_u Y_u \left\{ 1 - \prod_{v \neq u} \left(1 - Pr\left[u \sim v \mid u \in R,\ v \in L\right](1 - Y_v)\right) \right\} \right] \tag{39c}$$

$$\geq \mathbb{E}_Y \left[\sum_u Y_u \left\{ 1 - \prod_{v \neq u} \left(1 - q_{uv}\delta(v, B \mid u \in R, v \in L)(1 - Y_v)\right) \right\} \right] \tag{39d}$$

$$\geq \mathbb{E}_Y \left[\sum_u Y_u \left\{ 1 - \prod_{v \neq u} \left(1 - q_{uv}(1 - Y_v) \frac{1 - (1 - q_{uv}/\alpha) \prod_{w \neq u \neq v} (1 - q_{vw}Y_w/\alpha)}{\sum q_{vw}Y_w + q_{uv}} \right) \right\} \right] \tag{39e}$$

Equation (39c) follows from conditional probability, and equations (39d) and (39e) follow from Corollary 43, concerning our sampling technique. Since each of the random variables Y_u are chosen independently, (39e) can be simplified as below.

$$\mathbb{E}\left[\text{matched vertices in } \ R\right] \tag{40a}$$

$$\geq \sum_u \mathbb{E}_Y\left[Y_u\right] \mathbb{E}_Y \left[1 - \prod_{v \neq u} \left(1 - q_{uv}(1 - Y_v) \frac{1 - (1 - q_{uv}/\alpha) \prod_{w \neq u \neq v} (1 - q_{vw}Y_w/\alpha)}{\sum_{w \neq u \neq v} q_{vw}Y_w + q_{uv}} \right) \right] \tag{40b}$$

$$= \frac{1}{2} \sum_u \mathbb{E}_Y \left[1 - \prod_{v \neq u} \left(1 - q_{uv}(1 - Y_v) \frac{1 - (1 - q_{uv}/\alpha) \prod_{w \neq u \neq v} (1 - q_{vw}Y_w/\alpha)}{\sum_{w \neq u \neq v} q_{vw}Y_w + q_{uv}} \right) \right] \tag{40c}$$

In the following, (41b) can be derived from (40c) by using the identity $1 - x < e^{-x}$, and (41c) is obtained by noting that $(1 - 1/e)x < 1 - e^{-x}$ for $x \in [0, 1]$.

$$\mathbb{E}\left[\text{matched vertices in } \ R\right] \tag{41a}$$

$$\geq \frac{1}{2}\sum_u \mathbb{E}_Y\left[1-\exp\left(-\sum_{v\neq u} q_{uv}(1-Y_v)\frac{1-(1-q_{uv}/\alpha)\prod_{w\neq u\neq v}(1-q_{vw}Y_w/\alpha)}{\sum_{w\neq u\neq v} q_{vw}Y_w+q_{uv}}\right)\right] \tag{41b}$$

$$\geq \frac{1}{2}\sum_u \mathbb{E}_Y\left[\left(1-\frac{1}{e}\right)\left(\sum_{v\neq u} q_{uv}(1-Y_v)\frac{1-(1-q_{uv}/\alpha)\prod_{w\neq u\neq v}(1-q_{vw}Y_w/\alpha)}{\sum_{w\neq u\neq v} q_{vw}Y_w+q_{uv}}\right)\right] \tag{41c}$$

$$= \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_u \mathbb{E}_Y\left[\sum_{v\neq u} q_{uv}(1-Y_v)\frac{1-(1-q_{uv}/\alpha)\prod_{w\neq u\neq v}(1-q_{vw}Y_w/\alpha)}{\sum_{w\neq u\neq v} q_{vw}Y_w+q_{uv}}\right] \tag{41d}$$

Next observe that both $1-Y_v$ and $\frac{1-(1-q_{uv}/\alpha)\prod_{w\neq u\neq v}(1-q_{vw}Y_w/\alpha)}{\sum_{w\neq u\neq v} q_{vw}Y_w+q_{uv}}$ are decreasing convex functions in Y, thus their product is also a decreasing convex function. Our next set of simplifications are as follows. Since for any multi-variate convex function f, $\mathbb{E}[f(y)] \geq f(\mathbb{E}[y])$ we can lower bound equation (41d) by (42b) below. Again (42c) is minimized when each of the q_{vw}'s are equal. Substituting this and simplifying we get (42e). Finally $\frac{1-e^{-Q_v(G')/\alpha}}{Q_v(G')}$ is a decreasing function in $Q_v(G')$ that attains its minimum value when $Q_v(G')=1$. Putting this value in (42f) gives (42g). Further simplification yields the desired result.

$$\mathbb{E}\left[\text{matched vertices in } R\right] \tag{42a}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_u\sum_{v\neq u} q_{uv}(1-\mathbb{E}[Y_v])\frac{1-(1-q_{uv}/\alpha)\prod_{w\neq u\neq v}(1-q_{vw}\mathbb{E}[Y_w]/\alpha)}{\sum_{w\neq u\neq v} q_{vw}\mathbb{E}[Y_w]+q_{uv}} \tag{42b}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_{u}\sum_{v\neq u}\frac{q_{uv}}{2}\frac{1-(1-q_{uv}/\alpha)\prod_{w\neq u\neq v}(1-q_{vw}/2\alpha)}{\sum_{w\neq u\neq v}q_{vw}/2+q_{uv}} \tag{42c}$$

$$\approx \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_{u}\sum_{v\neq u}\frac{q_{uv}}{2}\frac{1-\prod_{w\neq v}(1-q_{vw}/2\alpha)}{\sum_{w\neq v}q_{vw}/2} \tag{42d}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_{u}\sum_{v\neq u}q_{uv}\frac{1-\exp\left(-\sum_{w\neq v}q_{vw}/2\alpha\right)}{\sum_{w\neq v}q_{vw}/2} \tag{42e}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_{u}\sum_{v\neq u}q_{uv}\frac{1-\exp\left(-Q_v(G')/2\alpha\right)}{Q_v(G')} \tag{42f}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\sum_{u}\sum_{v\neq u}q_{uv}\left(1-e^{-1/2\alpha}\right) \tag{42g}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\left(1-e^{-1/2\alpha}\right)\sum_{u}\sum_{v\neq u}q_{uv} \tag{42h}$$

$$\geq \frac{1}{2}\left(1-\frac{1}{e}\right)\left(1-e^{-1/2\alpha}\right)\sum_{u}Q_u(G') \tag{42i}$$

$$\geq \left(1-\frac{1}{e}\right)\left(1-e^{-1/2\alpha}\right)|OPT(G')|-\sum_{e}\epsilon_e\,. \tag{42j}$$

□

Let $\phi = (1-1/e)\left(1-e^{-1/2\alpha}\right)$. Let ALG_2 be the set of edges that are matched in the second stage of the algorithm. The following lemma lower bounds $\mathbb{E}[|ALG_2|]$.

Lemma 46.

$$\mathbb{E}[|ALG_2|] \geq \mathbb{E}[|OPT(G')|]\left[\frac{\phi}{1-(\frac{1-\phi}{2})^2}\right]-\sum_{e}\epsilon_e\,.$$

Proof. Observe that not all candidate edges in G' have been considered during the first iteration of Stage 2. In particular, candidate edges with both end points in R are yet to be considered. For analyzing the subsequent iterations in Stage 2, we will consider only these candidate edges. Clearly this only lower bounds the performance of the algorithm.

Analyzing equation (42i), we notice that we have in fact proved something stronger in Lemma 45, i.e., we have shown that every vertex $v \in R$ is chosen with probability at least $\phi\, Q_v(G')$. By slightly altering the algorithm it is easy to ensure that for every $v \in R$, it is chosen with exactly this probability. Thus any vertex in R survives the first iteration with probability $1 - \phi Q_v(G') > 1 - \phi$. Since the partitions L and R are chosen at random, the probability that a vertex is in R and unmatched after the first iteration is at least $\mu = (1-\phi)/2$. Continuing this argument further, the probability that an ordered pair (u, v) is a candidate edge at the start of the i^{th} iteration is the probability that both u and v have always been in R in all previous iterations, and are still unmatched; this probability is at least $\mu^{2(i-1)}$.

Let G'_i be the residual graph at the start of the i^{th} iteration in Stage 2, with $G'_1 = G'$. By the above observation and using linearity of expectation, the expected sum of q_e's on candidate edges in G'_i is lower bounded by $\mu^{2i-2} \sum_{e \in F(G')} q_e$. Appealing to a similar analysis as in (the proof of) Lemma 45, the expected size of the matching returned by the i^{th} iteration in the second stage is at least $\phi\mu^{2i-2} \sum_{e \in F(G')} q_e$. Summing over all iterations in Stage 2 we have,

$$\mathbb{E}[|ALG_2|] \geq \sum_i \phi\mu^{2i-2} \sum_{e \in F(G')} q_e \geq \phi \sum_{e \in F(G')} q_e \sum_{i=1} \mu^{2i-2} \tag{43a}$$

$$= \phi \sum_{e \in F(G')} q_e \frac{1}{1-\mu^2} \quad = \quad \frac{1}{2} \sum_{u \in G'} Q_u(G') \frac{\phi}{1-\mu^2} \tag{43b}$$

$$= |OPT(G')| \frac{\phi}{1-\mu^2} \quad = \quad |OPT(G')| \frac{\phi}{1 - \left(\frac{1-\phi}{2}\right)^2}\,. \tag{43c}$$

□

Now all that is left is to balance the factors for both the stages and set the optimal value of α. In the subsequent theorem we find the optimal value of α.

Theorem 47. *The above algorithm attains a factor of at least* $0.573 - 2\gamma$ *where* $\sum_e \epsilon_e \leq \gamma \mathbb{E}(|OPT|)$.

Proof. By Lemma 44, $\mathbb{E}[|OPT| - |OPT(G')|] \leq 2/(1+\alpha)\mathbb{E}[|ALG_1|] + \sum \epsilon_e$. Also by Lemma 46, $\mathbb{E}[|OPT(G')|] \leq \frac{1-(\frac{1-\phi}{2})^2}{\phi}\mathbb{E}[|ALG_2|] + \sum \epsilon_e$. Combining these two and substituting $\alpha = 0.255$ and $\phi = (1-1/e)\left(1-e^{-1/2\alpha}\right) = 0.543$ we have,

$$\mathbb{E}[|OPT|] \leq (2-\alpha)\mathbb{E}[|ALG_1|] + \frac{1-(\frac{1-\phi}{2})^2}{\phi}\mathbb{E}[|ALG_2|] + 2\sum_e \epsilon_e \tag{44a}$$

$$= 1.74(\mathbb{E}[|ALG_1|] + \mathbb{E}[|ALG_2|]) + 2\sum_e \epsilon_e \tag{44b}$$

$$= 1.74\mathbb{E}[|ALG|] + 2\sum_e \epsilon_e \tag{44c}$$

Thus $\mathbb{E}[|ALG|] \geq 0.573\ \mathbb{E}[|OPT|]$. □

4.4.1 Running Time Analysis

In this section we will analyze the running time of our algorithm and determine the optimal value of parameter C. We will use n to denote the number of vertices and m to denote the number of edges that have a non-zero probability of being present.

By Lemma 42 it takes $O(n^2)$ time to probe the neighborhood of a given vertex, thus the second stage can be implemented in $O(n^3)$ time. Analysis of the first stage is slightly involved, since in this stage we wish to approximate $q^*_{e,i}$ by repeated sampling. The following lemma sets the optimal value of C for which the total error caused by approximating $q^*_{e,i}$ by sampling is small.

Lemma 48. *For $C = n\log^6(n)$ samples in Step 1, $\sum_e \epsilon_e$ is $o(n)$ with high probability.*

Proof. We will give two separate bounds on the size of ϵ_e. One will hold in the case where $q^*_{e,i}$ is not too small, the other for small $q^*_{e,i}$.

Bound 1: For any given sample, we can think of the event $e \in M(H_j)$ as a Bernoulli trial with success probability $q^*_{e,i}$. By Hoeffding's bound ([29], see Theorem 1.8 in [65] for the specific formulation used), it follows that for any given edge and sample we have

$$\mathrm{P}(|q_{e,i} - q^*_{e,i}| \geq \beta q^*_{e,i}) \leq \exp(-Cq^*_{e,i}\zeta^2/4)\,.$$

This bound tells us that for any edge such that $Cq^*_{e,i}$ tends to infinity sufficiently quickly, the maximum error coming from such an edge will likely be a tiny fraction of $q^*_{e,i}$. However, it is possible that some q_e could be exponentially small, so we cannot just take C large enough so that all edges fall in this class. We turn to the second bound for the remaining edges.

Bound 2: In this case we focus solely on the upper tail. We know from the union bound that

$$\begin{aligned} \mathrm{P}(q_{e,i} \geq q^*_{e,i} + \kappa) &\leq \binom{C}{Cq^*_{e,i} + \kappa C} (q^*_{e,i})^{Cq^*_{e,i}+\kappa C} \\ &\leq \left(\frac{eq^*_{e,i}}{q^*_{e,i} + \kappa}\right)^{Cq^*_{e,i}+\kappa C}, \end{aligned}$$

where the first inequality bounds the probability that the edge participates in at least $C(q^*_{e,i}+\kappa)$ matchings by the expected number of sets of $C(q^*_{e,i}+\kappa)$ matchings in which the edge participates.

Set $q_0 = \log^5 n/C$ where we will use bound 1 for $q^*_{e,i} > q_0$ and bound 2 otherwise. For $q^*_{e,i} > q_0$ using bound 1, for an arbitrarily small constant ζ we have,

$$\mathrm{P}(|q_{e,i} - q^*_{e,i}| \geq \zeta q^*_{e,i}) \leq \exp(-\log^5 n\zeta^2/4) = o(1/n^4)\,.$$

Taking the union bound over all edges and trials and adding, we see

$$\mathrm{P}\Big(\sum_{q^*_{e,i} \geq q_0} \epsilon_e \leq \zeta \sum_e q^*_e\Big) = 1 - o(1)\,.$$

Thus the total error accrued across all iterations is small with high probability.

Now let us set $\kappa = 2q_0$. Applying the upper tail bound 2 above, we have for any $q^*_{e,i} < q_0$ that

$$\begin{aligned} \mathrm{P}(q_{e,i} \geq q^*_{e,i} + \kappa) &\leq \left(\frac{eq^*_{e,i}}{q^*_{e,i} + \kappa}\right)^{Cq^*_{e,i}+\kappa C} \\ &\leq (0.92)^{2\log^5 n}. \end{aligned}$$

The corresponding lower bound $q_{e,i} \geq q^*_{e,i} - \kappa$ follows trivially from the non-negativity of q_e.

Taking the union bound over all samplings and all edges, we have that with high probability the total contribution to the error from this case is at most $n^2\kappa = n^2 \log^5 n/C$. By Theorem 47, it suffices to make this a small fraction of the maximum expected matching. Setting $C = n \log^6 n$ ensures that the total error is $o(n)$ (which suffices in the case where the matching is a constant fraction of all vertices, while setting $C = n^2 \log^6 n$ insures the error is $o(1)$ (which works in general) [2] □

A naive implementation of the algorithm presented in section 4.3 would require recalculating q_e after every iteration in stage 1. This can be quite time consuming since even the fastest implementation [48] of the maximum matching algorithm takes $O(m\sqrt{n})$ time. In the following lemma, we explain how to circumvent this bottleneck. Finally, using Lemma 48 and 49 our algorithm can be implemented in $\tilde{O}(n^4)$ time.

Lemma 49. *Stage 1 of the algorithm can be implemented in $\tilde{O}(n^2C)$ time.*

Proof. Observe that we are not required to find the maximum matching in Step 1. We can instead work with a matching that is $1 - \zeta$ (ζ is an arbitrary small constant) fraction of maximum matching and lose a small multiplicative factor in our analysis. This can be done in $O(m \log m)$ time using a result by Duan and Pettie et al. [15]. Also note that we can reuse the above matching across multiple iterations in Stage 1. This is because the size of the maximum matching changes by at most 1 across consecutive iterations.

Concretely, we will modify the algorithm to calculate the approximate maximum matching using the algorithm in [15] in $O(m \log m)$ time. Let Λ be the estimate of the size of the maximum matching in the residual graph at any point during Stage 1.

[2] If the expected maximum matching is itself $o(1)$ in size, then it follows from the independence of the edges that any edge which is scanned and found to be present is with high probability the *only* edge in the graph! So any algorithm trivially finds the maximal matching in such a graph.

We can probe up to $\Lambda\zeta$ edges that have $q_{e,i}/p_e > \alpha$ before recalculating $q_{e,i}$. This will induce only a small constant factor (function of ζ) error in our analysis. Hence we would have at most $O(\log(m))$ iterations in Stage 1 where we would be required to recompute $q_{e,i}$. Therefore we can implement Stage 1 in $\tilde{O}(mC) = \tilde{O}(n^2C)$ time. □

4.5 Upper Bounds

In this section we will show an upper bound for any algorithm for SQCP by proving the following theorem.

Theorem 50. *No randomized algorithm can attain a competitive ratio better than* 0.896 *for the SQCP.*

Proof. Given any graph along with the probability p_e for every edge the optimal algorithm can be found by writing a stochastic dynamic program. The states of the program are all possible subgraphs of the given graph and for each state we record the solution returned by the optimal algorithm. It is easy to see that such a dynamic program would have exponentially many states and would be quite infeasible to solve for the general problem. However it can be used to find the optimal algorithm for small examples.

We considered the complete graph on 4 vertices where each edge is present with probability $p = 0.64$. In Lemma 51 we evaluate the performance of the optimal online algorithm for this graph. Then in Lemma 52 we calculate the expected size of the maximum matching in this graph.

Lemma 51. *The expected size of the matching found by the optimal online algorithm for SQCP for the Erdős-Rényi graph* $G(4, 0.64)$ *is* 1.607.

Proof. Let us consider the complete graph K_4 as shown in Figure 6(a). Without loss of generality we can assume that the optimal algorithm starts by scanning edge ac. If this edge is present, which happens with probability $p = 0.64$, then we are just left

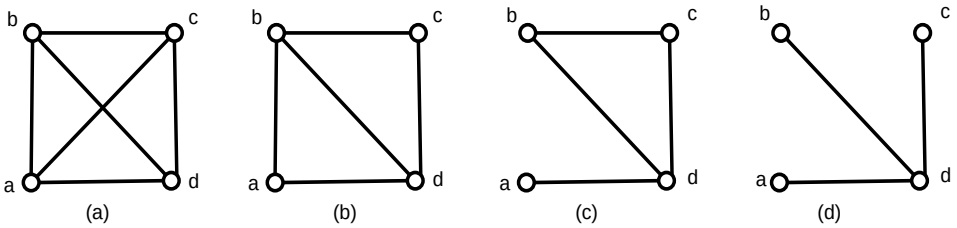

Figure 6: Intermediate Graphs

with one candidate edge. We can scan this edge next. Thus for this case the optimal algorithm return a matching of expected size $p + p^2$.

Now suppose that ac is not present, then we are left with the graph shown in Figure 6(b). Clearly the optimal algorithm should not probe edge bd since it can potentially lead to a smaller matching. Without loss of generality we may assume that the next edge to be scanned is ab. If this edge is present we are again left with one candidate edge cd that is to be scanned. Hence the expected size of the matching returned is $(1-p)p(1+p)$.

Otherwise we are left with the graph shown in Figure 6(c). One can check that the optimal algorithm must next scan bc or ad. Suppose it scans bc, and finds it to be present then we are again down to a single candidate ad which can be scanned next. In this the expected matching returned is of size $(1-p)^2p(1+p)$.

If bc is absent then the residual graph is shown in Figure 6(d). Clearly for the graph in Figure 6(d) the expected maximum matching is of size $1-(1-p)^3$. The expected size of the matching returned in this case is therefore $(1-p)^3(1-(1-p)^3)$. Computing the expected value across all cases and substituting $p = 0.64$ we find that the expected size of the matching returned by the optimal algorithm is 1.607.

□

Lemma 52. *The expected size of the maximum matching in the Erdős-Rényi graph* $G(4, 0.64)$ *is* 1.792

Proof. The given graph will not have any edge with probability $(1-p)^6$. It will have

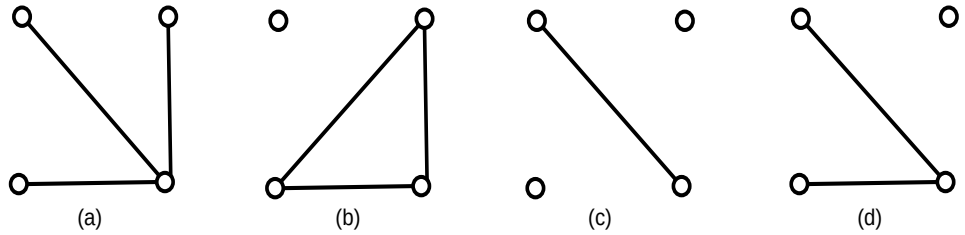

Figure 7: Graphs with unit size matching

a matching of size 1 for the graphs shown in figure 7 and their symmetric rotations.

This happens with probability $4p^3(1-p)^3+4p^3(1-p)^3+6p(1-p)^5+12p^2(1-p)^4$. In all other cases the graph will have a matching of size 2 i.e. with probability $1-(1-p)^6-8p^3(1-p)^3-6p(1-p)^5-12p^2(1-p)^4$. Thus the expected size of the maximum matching is $8p^3(1-p)^3+6p(1-p)^5+12p^2(1-p)^4+2(1-(1-p)^6-8p^3(1-p)^3-6p(1-p)^5-12p^2(1-p)^4)$. Substituting $p=0.64$ this is evaluates to 1.792. □

Combining the results of Lemma 51 and 52 we conclude that no online algorithm can achieve a factor better than $1.607/1.792 = 0.896$ for the SQCP. □

4.6 Combinatorial Algorithm for the Sampling Technique

In this section we will give a complete proof of Lemma 42 and also present a combinatorial algorithm for the sampling technique presented in Section 4.2.3.

By Lemma 41, we know that our constraint set is equivalent to

$$\sum_{i\in S} r_i + \prod_{i\in S}(1-p_i) \leq 1 \tag{45}$$

holding for every S. Suppose S^* is a non-empty set for which the left hand side of equation (45) is maximized, and there is some $j_1 \in S$ and $j_2 \notin S$. We would then have

$$\begin{aligned}
\sum_{i\in S^*} r_i + \prod_{i\in S^*}(1-p_i) &\geq \sum_{i\in S^*\setminus j_1} r_i + \prod_{i\in S^*\setminus j_1}(1-p_i) \\
\sum_{i\in S^*} r_i + \prod_{i\in S^*}(1-p_i) &\geq \sum_{i\in S^*\cup\{j_2\}} r_i + \prod_{i\in S^*\cup\{j_2\}}(1-p_i)
\end{aligned}$$

Rearranging both of the above inequalities yields

$$\begin{aligned}\prod_{i\in S^*}(1-p_i) &\leq \frac{r_{j_1}}{\frac{1}{1-p_{j_1}}-1}\\ \prod_{i\in S*}(1-p_i) &\geq \frac{r_{j_2}}{p_{j_2}}.\end{aligned}$$

Comparing these two inequalities, we have

$$\begin{aligned}\frac{r_{j_2}}{p_{j_2}} &\leq \frac{r_{j_1}}{\frac{1}{1-p_{j_1}}-1}\\ &= \frac{r_{j_1}(1-p_{j_1})}{p_{j_1}}\\ &\leq \frac{r_{j_1}}{p_{j_1}}.\end{aligned}$$

Assume without loss of generality that the events are sorted in decreasing order by the ratio r/p. By the above, we have proven

Claim 14. *The left hand side of equation* (45) *is always maximized by* $S = \{1, 2, \ldots, k\}$ *for some* k*.*

We also note the following:

Claim 15. *If equation* (45) *is satisfied for all* S *and tight for some* S_0*, then there is a solution to the program where any permutation having nonzero weight considers all variables in* S_0 *before any variable outside* S_0*.*

This is simply because the left hand side of equation (45) measures the sum of the probability that an event in S_0 is first and the probability that no event in S_0 occurs. If the sum is 1, then an event in S_0 must always be first if one is present, and that would not be possible if some other event could appear before it in π. We now present our algorithm.

Step 0: (pre-processing:) Compute the largest y for which the weights (yr_i, p_i) still satisfy equation (45). If $y < 1$, the constraints are infeasible. If $y > 1$, replace all r_i

by yr_i, at which point equation (45) is tight for at least one S. This preprocessing step is only performed once.

Step 1: (slack removal) If there is no $k < n$ for which equation (45) is tight for $S = \{1, 2, \ldots k\}$, compute the largest $z \leq 1$ such that the program remains feasible when for all j we replace r_j by

$$r'_j := \frac{r_j - zp_j \prod_{i>j}(1-p_i)}{1-z}.$$

With probability z, consider the edges in decreasing order of index. Otherwise, consider edges according to a distribution found by solving the problem with r_j replaced by r'_j. If $z = 1$, we are finished.

Step 2: (divide-and-conquer) Find a $k < n$ for which equation (45) is tight for $S = \{1, 2, \ldots, k\}$. Solve, without pre-processing, the problems on S (with the original target distribution) and S^C (replacing the targets in S^C by $r'_j := r_j / \prod_{i=1}^{k}(1-p_i)$). Form the distribution by independently sampling from the distributions on S and S^C found by solving the subproblems, and consider all variables in S before the variables in S^C.

For step 0, note that multiplication by y does not change the relative ordering of r_i/p_i. This implies that the y in question is the smallest y for which one of the sets $\{1, 2, \ldots, k\}$ makes equation (45) tight. We can examine all such sets and find the y in question in linear time by updating $\sum r_i$ and $\prod(1-p_i)$ each time k increases to $k+1$.

For step 1, note that

$$\begin{aligned}
\frac{r'_i}{p_i} - \frac{r'_{i+1}}{p_{i+1}} &= \frac{r_i - zp_i \prod_{j>i}(1-p_j)}{p_i} - \frac{r_i - zp_{i+1}\prod_{j>i+1}(1-p_j)}{p_{i+1}} \\
&= \left(\frac{r_i}{p_i} - \frac{r_{i+1}}{p_{i+1}}\right) + zp_i \prod_{j>i+1}(1-p_j) \\
&\geq 0.
\end{aligned}$$

In other words, replacing r_j by r'_j again never alters the relative ordering of the ratios. So again we only need consider n sets to determine z, and can do this in linear time.

The r'_j were chosen such that achieving a target of r'_j with probability $(1-z)$ corresponds to achieving a target of r_j in the original problem, so it is enough to solve this new problem. The only claim that remains to be checked is that we can actually find the desired k in Step 2. Since we know by our first claim that the left hand side of equation (45) is always maximized for some $S = \{1, 2, \ldots, k\}$, it suffices to show that we can take $k < n$.

But for $S = \{1, \ldots, n\}$, the left hand side of equation (45) is

$$\begin{aligned}
\sum_{j=1}^{n} r'_j + \prod_{j=1}^{n}(1-p_j) &= \frac{\sum_{j=1}^{n} r_j - z\sum_{j=1}^{n} p_j \prod_{i>j}(1-p_i)}{1-z} + \prod_{j=1}^{n}(1-p_j) \\
&= \frac{\sum_{j=1}^{n} r_j - z\left(1 - \prod_{j=1}^{n}(1-p_j)\right)}{1-z} + \prod_{j=1}^{n}(1-p_j) \\
&= \frac{\sum_{j=1}^{n} r_j + \prod_{j=1}^{n}(1-p_j) - z}{1-z} \\
&\leq 1,
\end{aligned}$$

where the last inequality comes from our assumption that equation (45) holds for $S = \{1, 2, \ldots, n\}$. Intuitively, this corresponds to how imposing constraints on the order in which the events are placed (increasing z) has no effect on the probability at least one event occurs (the left hand side of equation (45) in the case where S is everything). So at the maximal z (assuming $z < 1$), some other constraint must also be tight, which gives us our k.

Since step 1 takes at most linear time, it follows that the whole algorithm takes at most quadratic time.

4.7 Conclusion and Open Problems

In this chapter we discussed the stochastic relaxation of the query commit model presented earlier and gave a 0.575 factor algorithm based on a novel sampling technique (refer Section 4.2.3). We feel that apart from the SQCP this technique might be useful in several other realms of approximation algorithms. Finally we also show a

0.896 upper bound on the performance of any randomized algorithm for the stochastic query commit problem. Closing this gap is an interesting open problem.

CHAPTER V

ADWORDS IN THE STOCHASTIC QUERY COMMIT MODEL

5.1 Introduction

In this chapter we will consider a weighted generalization of the stochastic query commit problem introduced in Chapter 4. This is motivated by the session based online advertising. The online advertising model has three participants - (a) A search engine (b) Advertisers who wish to display ads for search queries; and finally (c) the users of the search engine.

Whenever the search engine receives a query from the user, it may choose to display ads along with the search results. Typically ad allocation is done by an online auction whereby the search engine solicits bids from the advertisers for every search query and the advertiser with the highest bid wins the right to show his ad for the query. In the cost-per-click model that is prevalent in the industry, the advertiser is charged the value of his bid only when the user clicks on his ad, after which he is redirected to the advertiser's website. Another feature of this model is the presence of daily budgets. The advertisers can set daily budgets to bound the maximum amount of money that they would spend in a day.

In the traditional model each search query is treated independently of the others even if it is issued by the same user. This may lead to unforeseen problems; for example a user may be shown the same ad (or similar ads) multiple times during a browsing session. Ideally the search engine should learn from the users choices during the search session to decide the best set of ads that would maximize the likelihood of the user clicking on an advertisement. This is difficult to implement in the current

model where each query is treated as an atomic entity.

A holistic approach towards addressing this problem is to introduce the notion of session based advertising where we analyze each browsing session to determine the best series of ads that must be shown to a user. This presents a novel set of challenges since unlike the traditional model where we only have to select one ad for every query, we are now required to choose a set of ads for every browsing session and show them over the course of the browsing session. Furthermore the search engine would like to maximize the money spent by the advertisers whilst ensuring that no advertiser spends more than his budget.

Towards this end we study the Generalized Assignment Problem (GAP) in the query commit model. In this model we are given a bipartite graph where the vertices on one side represent the advertisers, while the other bipartition represents the active users. Each vertex corresponding to an advertiser has a budget while each edge incident on it is weighted to reflect the amount he is willing to bid for a given user. Based on past user behavior we assume (Bayesian) priors on the edges indicating the probability that a given user would click on a chosen ad when presented with it during a search session. Finally we assume that if a user clicks on an ad he navigates away from the search engine and is lost forever. In this case the search engine charges the advertiser the smaller of his remaining budget and his bid for the user (weight of the corresponding edge). The objective for the search engine is to maximize the money spent by the advertisers.

5.1.1 Our Results

We study the GAP in the query commit model and compare our performance against an hypothetical adversary who knows the user preferences i.e. for every user the adversary knows the ads that are relevant to him. We also give this adversary infinite computational power to compute the optimal allocation using the knowledge of the

underlying graph. We prove the following results.

(a) There exists an algorithm that attains at least $1-1/e$ fraction of revenue earned by the adversary.

(b) For the case when the bids for every advertiser are much smaller than his budget the same algorithm that gets $1-o(1)$ fraction of revenue earned by the adversary.

5.1.2 Related Models and Results

Connections to the Adwords Problem: The Adwords problem was first introduced by Mehta et. al. in [47] where they studied an online version of this problem where the users arrive over time and each user makes a single query. They further assumed that the users arrive in a fixed adversarially chosen order and all bids are very small with respect to the budgets. For this problem they showed a tight $1-1/e$ factor deterministic algorithm. Later, Buchbinder et al. [8] gave a primal dual algorithm for the same problem.

Understanding the adwords allocation problem in distributional models was an posed as an open question in [43, 47]. This was addressed by Devanur and Hayes in [14] where they studied the adwords problem in the unknown distribution model similar to the one presented in this Chapter 2 and obtained a $1-o(1)$ competitive algorithm for this problem.

We would like to point out that in general the adwords problem with bids much smaller than the budgets is incomparable to graph matching: It is easier in the sense of having large budgets (hence mistakes can be rectified in the future), but harder because of different bid values.

Connections to the GAP: The GAP has been part of operations research and computer science folklore for several years. It was shown to be NP-hard by Lehmann et. al in [41] where they also presented a simple $1-1/e$ factor algorithm for this problem. This factor was later improved by Chakrabarty and Goel [10] to 0.75,

matching the integrality gap of the natural linear programming relaxation.

The problem is also a special case of the submodular welfare maximization (SWM) problem first studied by Wolsey in [68], where he showed that a simple greedy algorithm attains a factor of 0.5. In [38] Khot et. al. showed that SWM is hard to approximate to a factor better than $1 - 1/e$ and its approximability was finally settled by Vondrak in [67] where he gave an optimal continuous greedy algorithm.

5.2 Preliminaries

5.2.1 Problem Statement

We are given the vertex set $L \cup R$ of a bipartite graph G. The vertices in L represent advertisers while those in R represent active users. For each $v \in L$ we are given a budget $B_v > 0$. The edges in the graph are not revealed to us, but for every *candidate edge* uv, we are told the probability (denoted by p_{uv}) that it is present. For every edge uv we are also given a weight f_{uv} denoting the advertiser's bid.

We can scan/probe the edges in the graph to check if they are present. Whenever an edge uv is scanned (i.e. advertiser v shows an ad to user u) one of two things may happen - Either the edge is found to be present (the ad was relevant to the user) or the edge is absent in the underlying graph (the user chose to ignore the ad). In the former case we charge the advertiser v the smaller of his bid and his remaining budget. The user u is redirected to the advertiser's website and plays no further part in the algorithm. In this case the ad-impression is said to be *assigned* to the user. For the latter case, we simply note that the edge uv is absent; the user remains active except that he is not shown any more ads from advertiser v.

The objective is to maximize the total money (also called revenue) that is charged to the advertisers. We compare our performance against an adversary $\mathcal{J}$ who knows the underlying graph for every instantiation of the problem. As discussed in Section 5.1.2 even with complete knowledge of the underlying graph the given problem boils

down the GAP which is NP-hard. Thus we also give the adversary $\mathcal{J}$ unbounded computational power to find the optimal ad-allocation in the underlying graph.

We say an online algorithm $\mathcal{A}$ attains a competitive ratio of γ, if for every problem instance $\mathcal{I} = (G(L \cup R, .), p)$, the expected revenue returned by $\mathcal{A}$ is at least γ times the expected maximum revenue attained by the omniscient computationally unbounded adversary $\mathcal{J}$ i.e. $\gamma = \min_{\mathcal{I}=(G(V,.),p)} \left\{ \frac{E\left[\mathcal{A}(\mathcal{I})\right]}{E\left[\text{ max. revenue found by } \mathcal{J} \text{ for } G(L \cup R, p)\right]} \right\}$.

5.2.2 Properties of the Optimal Solution

There are two obstacles in developing a good understanding of the optimal solution. The first problem arises due to the stochastic nature of the input while the second difficulty is due to the inherent hardness of the underlying optimization problem (GAP). We tackle both these concerns by relaxing the above problem to the following linear program.

$$\begin{array}{lll} \max & \sum_{u \in R} \sum_{v \in L} x_{uv} f_{uv} & (\text{LP}_{AW}) \\ & \sum_{v \in L} x_{uv} f_{uv} \leq 1 & \forall u \in R \\ & \sum_{v \in L} x_{uv} \leq 1 - \prod_{v \in S} (1 - p_{uv}) & \forall u \in R,\ S \subseteq L \\ & \sum_{u \in R} x_{uv} f_{uv} \leq B_v & \forall v \in L \\ & 0 \leq x_{uv} \leq 1 & \forall u \in R,\ v \in L \end{array}$$

Here x_{uv} represents whether user u clicked on the ad shown by advertiser v. Note that this encompasses both the events - v's ad was shown to u and it was also relevant to him. Therefore $x_{uv} \leq p_{uv}$. The second set of constraints is a generalization of this inequality to arbitrary sets of advertisers. It is easy to see that the first constraint states that every user clicks on at most one ad in a browsing session, while the third constraint asserts that no advertiser overshoots his budget.

Let us define $x^*_{uv} = \Pr\left[uv \text{ is chosen by } \mathcal{J}\right]$, where the probability is taken over random bits in the distribution over the edges. It is easy to see that x^* is a feasible

solution to LP_{AW}. Thus we have the following observation.

Observation 3. *The optimal solution to LP_{AW} is an upper bound on the expected revenue earned by the adversary $\mathcal{J}$.*

5.3 Algorithm for the Adwords in the Query Commit Model

In this section we will explain our algorithm for the Adwords problem. Our algorithm can be divided in to two stages.

Stage 1: In the first stage we solve LP_{AW}. Since LP_{AW} has exponentially many constraints we need a separating oracle to find an optimal solution. In Lemma 53 we explain how to build this separating oracle.

Lemma 53. *There exists a polynomial time separating oracle for LP_{AW}.*

Proof. Given any solution x the first and third and fourth set of constraints in LP_{AW} are easy to check. The algorithm for testing the second set of constraints is derived from the proof of Claim 14 in Chapter 4.

In the second set of constraints we need to verify that for every $u \in R$

$$\sum_{v \in S} x_{uv} + \prod_{v \in S}(1 - p_{uv}) \leq 1 \quad \forall S \subseteq L \tag{46}$$

For the rest of the proof let us fix $u \in R$. Suppose S^* is a non-empty set for which the left hand side of equation (46) is maximized. Then equation (46) is violated iff $\sum_{v \in S^*} x_{uv} + \prod_{v \in S^*}(1 - p_{uv}) > 1$. If there is some $v_1 \in S^*$ and $v_2 \notin S^*$ then we would have

$$\begin{aligned}
\sum_{v \in S^*} x_{uv} + \prod_{v \in S^*}(1 - p_{uv}) &\geq \sum_{v \in S^* \setminus v_1} x_{uv} + \prod_{v \in S^* \setminus v_1}(1 - p_{uv}) \\
\sum_{v \in S^*} x_{uv} + \prod_{v \in S^*}(1 - p_{uv}) &\geq \sum_{v \in S^* \cup \{v_2\}} x_{uv} + \prod_{v \in S^* \cup \{v_2\}}(1 - p_{uv})
\end{aligned}$$

Rearranging both of the above inequalities yields

$$\prod_{v \in S^*}(1 - p_{uv}) \leq \frac{x_{uv_1}}{\frac{1}{1 - p_{uv_1}} - 1}$$

$$\prod_{v \in S^*} (1 - p_{uv}) \quad \geq \quad \frac{x_{uv_2}}{p_{uv_2}}$$

Comparing these two inequalities, we have

$$\begin{aligned} \frac{x_{uv_2}}{p_{uv_2}} &\leq \frac{x_{uv_1}}{\frac{1}{1-p_{uv_1}} - 1} \\ &= \frac{x_{uv_1}(1 - p_{uv_1})}{p_{uv_1}} \\ &\leq \frac{x_{uv_1}}{p_{uv_1}}. \end{aligned}$$

Thus if we sort the edges in the set $\{uv \mid v \in L\}$ in decreasing order of the ratio x_{uv}/p_{uv} then we only need to check sets that are prefixes of this ordering to verify equation (46). This can be easily accomplished in polynomial time. □

Armed with the above separating oracle we can find an optimal solution $\bar{x}$ to LP_{AW} and are now ready to commence the second stage of the algorithm.

Stage 2: Here we use the optimal fractional solution together with the sampling technique described in Section 4.2.3 to decide the order in which the edges should be scanned. Recall the sampling lemma from Section 4.2.3

Lemma 54. *Let $A_1, A_2, \ldots A_k$ be independent events having probabilities $p_1, \ldots, p_k$. Let $r_1, \ldots, r_k$ be fixed non-negative constants such that for every $S \subseteq \{1, \ldots, k\}$ we have*

$$\sum_{i \in S} r_i \leq 1 - \prod_{i \in S} (1 - p_i). \tag{47}$$

Then there is a probability distribution over permutations π of $\{1, 2, \ldots, k\}$ such that for each i, we have

$$\mathrm{P}(A_i \text{ is the earliest occurring event in } \pi) \geq r_i \tag{48}$$

Furthermore this distribution can be found in polynomial time.

Note that for any $u \in L$ the set of variables $\{\bar{x}_{uv} \mid v \in R\}$ satisfy the requirements for Lemma 54. This follows from the third set of constraints in LP_{AW}. In the second

stage we iterate through the vertices in L and for each vertex u we use Lemma 54 to decide the order in which to scan the edges incident on it. We stop when we have processed all vertices in L.

5.4 Analysis of the Sampling Based Algorithm

In this section we will analyze the performance of the above algorithm. Recall that in the first stage of the algorithm we found the optimal fractional solution $\bar{x}$ to LP_{AW}. For every edge uv, let us associate a Bernoulli random variable X_{uv} that is 1 with probability $\bar{x}_{uv}$ and 0 otherwise. Also, let $\mathcal{A}_v$ be v's expected contribution to the revenue. We computer $\mathcal{A}_v$ in terms of X_{uv} in Lemma 55.

Lemma 55. *For any vertex $v \in L$,*

$$\mathcal{A}_v = \mathbb{E}\left[min\left\{\sum_{u\in L} X_{uv} f_{uv}\ , B_v\right\}\right]$$

Proof. By linearity of expectation $\mathcal{A}_v = \min\left\{\sum_{u\in L} \Pr\left[u \text{ is assigned to } v\right] f_{uv}\ , B_v\right\}$ Since we use the technique described in Lemma 54 to scan the edges, any edge uv gets assigned with probability at least $\bar{x}_{uv}$, therefore $\mathcal{A}_v = \mathbb{E}\left[\min\left\{\sum_{u\in L} X_{uv} f_{uv}\ , B_v\right\}\right]$. □

Also note that v's expected contribution the revenue in the optimal solution is given by $\mathcal{J}_v = \min\left\{\sum_{u\in R} \bar{x}_{uv} f_{uv}, B_v\right\}$. By the third set of constraints in LP_{AW} this is equal to $\sum_{u\in R} \bar{x}_{uv} f_{uv}$. Thus we can lower bound the approximation factor for our algorithm by $\gamma \geq \min_{v\in L}\left\{\frac{\mathcal{A}_v}{\mathcal{J}_v}\right\} = \min_{v\in L}\left\{\frac{\mathbb{E}\left[min\left\{\sum_{u\in L} X_{uv} f_{uv}\ , B_v\right\}\right]}{\sum_{u\in R} \bar{x}_{uv} f_{uv}}\right\}$. Define another Bernoulli random varaible Y_{uv} that is 1 with probability $\bar{x}_{uv} f_{uv}/B_v$. We make the following observation based on the above definitions.

Observation 4. *For every $v \in L$, $\mathbb{E}\left[\sum_{u\in R} X_{uv} f_{uv}\right] = \mathbb{E}\left[\sum_{u\in R} B_v Y_{uv}\right]$*

Lemma 56 is a consequence of Observation 4 stated above.

Lemma 56. *For every $v \in L$, $\mathbb{E}\left[\min\left\{\sum_{u\in R} X_{uv} f_{uv}, B_v\right\}\right] \geq \mathbb{E}\left[\min\left\{\sum_{u\in R} B_v Y_{uv}, B_v\right\}\right]$*

Proof. The lemma follows from the fact that both random variables Y_{uv} and $X_{uv}f_{uv}/B_v$ have the same expectation but Y_{uv} happens with smaller probability. □

Next, in Lemma 57 we use Lemma 56 to show that γ is at least $1 - 1/e$. Then in Lemma 58 we prove that $\gamma > 1 - o(1)$ under the assumption that bids (f_{uv}) are much smaller than the budgets (B_v).

Lemma 57. $\gamma \geq 1 - 1/e$

Proof.

$$\begin{aligned}
\gamma &\geq \min_{v \in L} \left\{ \frac{\mathbb{E}\left[\min\left\{\sum_{u \in R} Y_{uv}, 1\right\}\right] B_v}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} && (49) \\
&\geq \min_{v \in L} \left\{ \frac{\left(1 - \prod_{u \in R}(1 - Y_{uv})\right) B_v}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} && (50) \\
&\geq \min_{v \in L} \left\{ \frac{\left(1 - exp(-\sum_{u \in R} Y_{uv})\right) B_v}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} && (51)
\end{aligned}$$

Here, equation (49) follows from Lemma 56. Now two cases may arise.

Case 1: $\sum_{v \in L} Y_{uv} \leq 1$

Using the identity $1 - e^{-z} \geq z(1 - 1/e)$ for $z \in [0, 1]$ in equation (51) ,

$$\begin{aligned}
\gamma &\geq \min_{v \in L} \left\{ \frac{(1 - 1/e) \sum_{u \in R} B_v Y_u v}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} \\
&= \min_{v \in L} \left\{ \frac{(1 - 1/e) \sum_{u \in R} X_{uv} f_{uv}}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} \\
&= 1 - 1/e
\end{aligned}$$

Case 2: $\sum_{v \in L} Y_{uv} \geq 1$

From equation (51) ,

$$\begin{aligned}
\gamma &\geq \min_{v \in L} \left\{ \frac{(1 - 1/e) B_v}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} \\
&\geq 1 - 1/e
\end{aligned}$$

□

Lemma 58. *If every f_{uv} is much smaller than B_v then, $\gamma \geq 1 - o(1)$*

Proof. Recall that for any $v \in L$, $\mathbb{E}\left[\sum_{u \in R} Y_{uv} B_v\right] = \sum_{u \in R} \bar{x}_{uv} f_{uv} \leq B_v$. Thus $\mathbb{E}\left[\sum_{u \in R} Y_{uv}\right] \leq 1$. Since each Y_{uv} is much smaller than 1, by Chernoff bounds, the expected value of $\sum_{u \in R} Y_{uv}$ is tightly concentrated around its mean. Therefore, $\mathbb{E}\left[\min\left\{\sum_{u \in R} Y_{uv}, 1\right\}\right] > 1 - o(1)$. Hence from equation (49)

$$\begin{aligned} \gamma &\geq \min_{v \in L} \left\{ \frac{(1 - o(1))\, B_v}{\sum_{u \in R} \bar{x}_{uv} f_{uv}} \right\} \\ &\geq 1 - o(1) \end{aligned}$$

□

The results from Lemmas 57 and 58 are summarized in the following theorem.

Theorem 59. *The algorithm described in Section 5.3 attains at least $1 - 1/e$ fraction of the optimal revenue. For the case when the bids for every advertiser are much smaller than his budget the same algorithm that gets $1 - o(1)$ fraction of optimal revenue.*

5.5 Conclusion and Open Problems

In this chapter we studied the adwords problem in the stochastic query commit model and gave an algorithm that attains a factor of $1 - 1/e$. We also showed that the performance of the same algorithm improves significantly if the bids are much smaller than the budget.

A major drawback of the model considered in this chapter is that it assumes that the set of users is fixed/static. In reality users may arrive and depart throughout the day. Understanding the online version of the session based advertising model studied here remains an interesting avenue of future research.

CHAPTER VI

ALLOCATION PROBLEMS WITH ECONOMIES OF SCALE

6.1 Introduction

In the previous chapters we considered several allocation problems with linear objective functions. However linear functions do not always model the complex dependencies that exist in the real world. Motivated by these concerns we will consider a more general class of valuation functions called submodular functions. Another feature that arises in practice is the presence of multiple agents, where each agent has her own cost function. For linear cost functions this does not present any added difficulty but this is not the case for general cost functions. In this chapter we will study the limits of approximability two fundamental allocation problems with multiple agents and non-linear cost functions.

6.1.1 Problem Statement

We consider the situation where there are multiple agents who wish to collectively perform a certain task, such as building a large combinatorial structure at the minimum possible total cost. Formally, we are given a set of elements X and a collection $C \subseteq 2^X$. We are also given m agents, where each agent i specifies a cost function $f_i : 2^X \to R^+$. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_m$ of S such that $\sum_i f_i(S_i)$ is minimized. We will consider the following general classes of functions to model the non-linearity of the agents' costs.

Submodular Cost Functions: Submodular functions form a rich class and capture natural properties of economies of scale and the law of diminishing returns. A function

$f : 2^X \to R^+$ is said to be submodular iff for any two sets S and $T \subseteq X$, $f(S)+f(T) \geq f(S \cup T)+f(S \cap T)$. Function f is said to be monotone if $f(S) \leq f(T)$ for any $S \subseteq T$, and normalized if $f(\emptyset) = 0$. Note that linear functions, i.e. functions of the form $f(S) = \sum_{i \in S} a_i$, are a special and classically studied sub case of submodular functions, Since a submodular function is defined over an exponentially large domain, we will work with the *value oracle* model in which an oracle will return the value of $f(S)$, when queried with the set $S \subseteq X$.

Discounted Cost Functions: Unfortunately, many of the assignment problems that we study turn out to be extremely hard under submodular functions [23, 27, 30, 63]. Thus, from a practical standpoint, the applicability of these results is not well-founded as the class of submodular functions might too general. Moreover, the class of submodular functions is defined over an exponentially large domain and thus requires exponential time to write down the function explicitly which may not be the case in real-world applications.

Thus, we wish to explore functions that lie between the additive functions and the general submodular functions, and that are also succinctly representable. In particular, we study *discounted cost functions* - We are given an additive cost function $c : X \to R^+$ and a discount function $d : R^+ \to R^+$ that satisfies the following properties: (1) $d(0) = 0$; (2) d is increasing; (3) $d(x) \leq x$ for all x; (4) d is concave. The price of any $S \subseteq X$ is defined to be $d(c(S))$. It is not difficult to see that discounted price functions form a subclass of submodular functions.

6.1.2 Our Results

We study the following problems in this paradigm for both types of cost functions described above.

1. **Combinatorial Reverse Auction (CRA):** We are given a set X of n elements and we wish to partition this set among m agents to minimize the total

cost incurred by the agents.

2. **Minimum Cost Perfect Matching:** We are given an undirected graph $G(V, E)$ and the agents' cost functions are over subsets of edges. We are further told that G has at least one perfect matching. The objective is to find a perfect matching in G and allocate the edges among the agents so as to minimize the total cost.

Our results are summarized in Table 3 below. We would like to draw attention to the fact that the lower bounds for submodular cost functions are information theoretic (refer to Section 6.2.1 for details) while those for discounted cost functions are contingent on P $\neq$ NP.

Table 3: Summary of Results for Allocation Problems with Non-linear Cost Functions

	Submodular Cost Functions		Discounted Cost Functions	
	Lower bound	Upper bound	Lower bound	Upper bound
Reverse Auction	$\Omega(\log n)$	$\min(m, \log n)$	$\Omega(\log n)$[19]	$\min(m, \log n)$
Perfect Matching	$\Omega(n)$	n	$\Omega(polylog(n))$	n

6.1.3 Related Models and Results

Submodular functions have been intensely studied from the optimization perspective in the past. Perhaps, the most fundamental optimization problem concerning submodular functions is the non-monotone submodular function minimization problem. A sequence of papers in this direction [31, 32, 33, 34, 51, 59] has resulted in fast strongly polynomial time combinatorial algorithms. On the other hand the related problem of maximizing non-monotone submodular function is known to be NP-hard and the best known constant factor algorithm for this problem is by Feige et. al. [20].

Another body of work in optimization over submodular functions deals with welfare maximization [9, 21, 38, 67]. In this context, the reverse auction problem (CRA)

that we study, can be thought of as submodular welfare minimization. Calinescu et al. [9] studied submodular function maximization subject to matroid constraints. They showed that their problem contains many other allocation problems as sub cases, thus giving a unified framework for studying such problems. Matching information theoretic lower bounds were established in [49].

Very recently, Svitkina and Fleischer [63] studied submodular objective function for problems like sparsest cut, load balancing, and knapsack. They gave $O\left(\sqrt{\frac{n}{\log n}}\right)$ upper and lower bounds for all these problems, showing that all these problems become much harder under submodular costs. Later Iwata and Nagano [30] studied the vertex cover, edge cover and set cover under submodular functions.

For the submodular reverse auctions, where a set of n goods has to be allocated to m agents (i.e. collection set $C = \{X\}$) with submodular cost functions to minimize the overall cost, a simple greedy algorithm is known to have a factor $\log(n)$ [28]. Goemans et al. [27] gave an algorithm for constructing explicit approximate submodular functions by querying polynomial number of times to the original submodular function. It is interesting to note that the construction in [27] used to approximate a given submodular function belongs to the class of discounted cost functions defined earlier. Refer to [28, 60, 62, 64, 68] for some other related work in optimization that uses submodular functions. .

6.2 Submodular Cost Functions

In this section we will consider the case when the agents' cost functions are monotone submodular functions. Recall that these functions are defined over all subsets of a given ground set. We therefore assume value oracle access to the functions. We will now define the notion of an information theoretic lower bound and develop a general technique to derive such bounds.

6.2.1 Information Theoretic Lower Bounds

A problem is said to have information theoretic lower bound of α if any randomized algorithm that approximates the optimum to a factor α with high probability requires super-polynomial number of queries to the value oracle.

By Yao's principle [69], it suffices to establish the lower bounds for deterministic algorithms acting on an input which is picked randomly from some fixed distribution. To show these approximation gaps, we follow the general framework which was also used in [20, 27, 63]. We will outline this framework in the single agent setting.

The idea is to first choose a problem instance which has a suitably large collection set $C \subseteq 2^X$ of interest. For example, for the minimum perfect matching problem, we choose a graph that has exponentially many perfect matchings. Then we design two submodular cost functions f and g. Typically, g is deterministically picked, whereas f is chosen from a distribution. The choice of f and g relies on the following two properties: a) The optimum values of f and g over C must differ by a large factor, and b) f and g must be 'hard to distinguish' in the sense that, for any deterministic query $Q \subseteq X$, both functions return the same value with a high probability.

By the union bound and a computation path argument [20, 63], a deterministic algorithm making polynomially many number of queries cannot distinguish between f and g. Combining this with the gap in the optima of f and g, one proves the lower bound. We summarize the above discussion in the following observation.

Observation 5. *To prove an information theoretic lower bound using two two-partition functions f and g with a gap in their optimum values, it suffices to prove that, for an arbitrary subset $Q \subseteq X$, $Pr[\ f_(Q) \neq\ g(Q)\ ]$ is super-polynomially small over the random choice of subset $R \subseteq X$.*

6.2.2 Combinatorial Reverse Auctions

In this problem we are given a set X, of n elements and m agents. For each agent i we have a normalized monotone submodular cost function $f_i : 2^X \to \mathbb{R}^+$. We wish to partition the elements among the agents to minimize the total cost. We prove a $\Omega(\log n)$ information theoretic hardness result and provide an algorithm that matches this bound. We also prove the same algorithm to be m-approximate.

6.2.2.1 ***Proof of hardness***

As discussed in Section 6.2.1, the idea is to construct a deterministic instance and a random instance of the CRA so that the optimal solution of these two instances differ by a factor of $\Omega(\log n)$, and then show that with high probability, a deterministic algorithm which uses only polynomially many value queries can not distinguish between these two instances.

The deterministic instance we will use is the following: There are m agents and a set X of $n = m(m+1)^2/4$ elements. The elements are equally partitioned into m disjoint sets $X_1, X_2, \cdots, X_m$. We will choose m such that $m = 2^d - 1$, for some d. Now each number i between 1 and m can be represented as a vector a_i in $GF[2]^d$. Let $G_i = \bigcup_{1 \le k \le m,\ a_i \cdot a_k = 1} X_k$. For each i, $1 \le i \le m$, agent i is only interested in elements in G_i. It is easy to see that G_i consists of exactly $\frac{m+1}{2}$ blocks and for each block there are $(m+1)/2$ agents who are interested in it. Now, we define the cost function $g_i : 2^X \longrightarrow \mathbb{R}^+$ as follows:

$$g_i(S) \;=\; \begin{cases} \min\{|S|, (m+1)^2/4\} & \text{If } S \subseteq G_i \\ \infty & \text{Otherwise} \end{cases}$$

Let us analyze the optimal cost of this instance. We say that an agent is *marked* if the total size of elements assigned to him is at least $(m+1)^2/4$. Among all the optimal solutions, let OPT be the one that maximizes the number of marked agents.

We claim that at least d agents are marked in OPT. Suppose not, then without loss of generality, we may assume $M = \{1, 2, ..., t\}$ to be the set of marked agents and $t < d$. The system of linear equations $a_i \cdot x = 0, \forall 1 \leq i \leq t$ has at least one solution $x^* \in GF[2]^d$, since number of equations is less than the number of variables. Let k be the number between 1 and m corresponding to the vector x^*. This implies that no agent in M is interested in block X_k. Let $A_k = \{i_1, i_2, ..., i_w\}$ be the set of agents who are assigned elements from X_k. Then $A_k \cap M = \emptyset$. Therefore, we can mark one more agent by transferring the elements in X_k from agents $i_2, i_2, ..., i_w$ to agent i_1 without changing the cost of the new solution. This is a contradiction because of the choice of OPT. Hence, the optimal cost of this instance is at least $(m+1)^2 d/4$.

Next we will describe the randomized instance which has same the set of agents and elements as the deterministic instance. Also, each agent is interested in the same set of elements. However, the cost function for each agent is picked in a randomized manner. We describe in detail below.

For each element, assign it uniformly at random to one of the agents who is interested in it. Let S_i be the set of elements which agent i gets. Clearly $(S_1, S_2, ..., S_m)$ forms a partition of the element set X. We define the cost function $f_i : 2^X \longrightarrow \mathbb{R}^+$, for agent i as follows:

$$f_i(S) = \begin{cases} \min\left(\ |S \cap \overline{S_i}| + \min\left\{|S \cap S_i|, (1+\delta)(m+1)/2 \ \right\}, (m+1)^2/4\right) & \text{If } S \subseteq G_i \\ \infty & \text{Otherwise} \end{cases}$$

where $\delta > 0$ is a fixed constant.

Now we show that with high probability, a deterministic algorithm using only polynomially many value queries can not distinguish between $f = (f_1, f_2, ..., f_m)$ and $g = (g_1, g_2, ..., g_m)$. We prove the following lemma.

Lemma 60. *For any subset S of elements and $i \in [m]$, $Pr[f_i(S) \neq g_i(S)] = e^{-\Omega(m)}$.*

Proof. Suppose S is a subset of elements and $i \in [m]$. Using our construction we have $f_i(S) \leq g_i(S)$. Therefore $Pr[f_i(S) \neq g_i(S)] = Pr[f_i(S) < g_i(S)]$.

First of all, we claim that the above probability is maximized when $S \subseteq G_i$ and $|S| = (m+1)^2/4$. For this, if $S \not\subseteq G_i$, then $f_i(S) = g_i(S) = \infty$ hence $Pr[f_i(S) < g_i(S)] = 0$. Now suppose $S \subseteq G_i$ and $|S| \geq (m+1)^2/4$. Then $g_i(S) = (m+1)^2/4$. Therefore

$$Pr\left[f_i(S) < g_i(S)\right] \;=\; Pr\left[|S \cap \overline{S_i}| + \min\{|S \cap S_i|, (1+\delta)(m+1)/2\} \;<\; (m+1)^2/4\right]$$

This probability can only increase when we remove elements from S. For the case when $|S| \leq (m+1)^2/4$, we get:

$$\begin{aligned} Pr\left[\; f_i(S) < g_i(S) \;\right] &= Pr\left[\; |S \cap \overline{S_i}| + \min\{|S \cap S_i|,\; (1+\delta)(m+1)/2 \;\} \;<\; |S| \;\right] \\ &= Pr\left[\; \min\{|S \cap S_i|,\; (1+\delta)(m+1)/2 \;\} \;<\; |S \cap S_i| \;\right] \\ &= Pr\left[\; |S \cap S_i| \;>\; (1+\delta)(m+1)/2 \;\right] \end{aligned}$$

Thus, this probability can only increase when more elements are added to S. Hence under the condition $S \subseteq G_i$, $|S| \leq (m+1)^2/4$, the probability is also maximized when $|S| = (m+1)^2/4$.

Now we assume $S \subseteq G_i$ and $|S| = (m+1)^2/4$. In this case, $Pr[f_i(S) < g_i(S)] = Pr[|S \cap S_i| > (1+\delta)(m+1)/2]$, which by a standard Chernoff bound arguments, can be shown to be bounded by $e^{-\Omega(m)}$. □

If we define $f(S) = (f_1(S), ..., f_m(S))$ and $g(S) = (g_1(S), ..., g_m(S))$, then by a simple union bound, as a corollary of the lemma, we have $Pr[f(S) \neq g(S)] = poly(m)e^{-\Omega(m)}$. Now suppose $\mathcal{A}$ is a deterministic algorithm which makes polynomially many queries to the value oracle. Then by the union bound, with probability at most $poly(m) \cdot e^{-\Omega(m)}$, $\mathcal{A}$ can distinguish between f and g. Notice that for the cost function $f = (f_1, \cdots, f_m)$, the optimal solution is at most $(1+\delta)m(m+1)/2$ achieved by assigning S_i to agent i. However, as we showed, the optimal solution for the cost

function $g = (g_1, g_2, \cdots, g_m)$ has cost at least $d(m+1)^2/4$, thus with high probability, $\mathcal{A}$ can not approximate a CRA instance within factor $\frac{(m+1)^2 d/4}{(1+\delta)m(m+1)/2} \simeq d = c \log n$ for some $c < 1$. Finally, by Yao's principle, we have the following:

Theorem 61. *A randomized approximation algorithm for the CRA problem within factor $c \log n$ for some $c < 1$ needs to make exponentially many value queries.*

6.2.2.2 ***A $\min(m, \log n)$ approximation algorithm***

A $\log n$-approximate algorithm for this problem appeared in [28]. In what follows we provide a $\min(m, \log n)$ approximation algorithm. Consider the following LP relaxation (LP1) and its dual (LP2).

$$\begin{array}{llll} \min \sum_{S \subseteq V} \sum_i x_{i,S} f_i(S) & \text{(LP1)} & \max \sum_{u \in X} y_u & \text{(LP2)} \\ \sum_{S: u \in S} \sum_i x_{i,S} \geq 1 & \forall u \in X & \sum_{u \in S} y_u \leq f_i(S) & \forall S \subseteq V,\ \forall i \\ x_{i,S} \geq 0 & \forall S \subseteq V,\ \forall i & y_u \geq 0 & \forall u \in X \end{array}$$

In LP1, $x_{i,S}$ is used to represent the fraction of set S that is allocated to agent i. Since $f_i(S) - \sum_{u \in S} y_u$ is a submodular function, we can construct a separation oracle for the dual program using the submodular minimization algorithm as a subroutine. Thus we can solve LP1 and LP2 optimally. The following lemma describes the structure of an optimal solution to LP1.

Lemma 62. *There exists an optimal fractional solution to LP1 such that for every agent i the set $\mathcal{T}_i = \{\ S\ :\ x_{i,S} > 0\ \}$ forms a nested family.*

Proof. Let x be any feasible solution to LP1. If $\mathcal{T}_i$ is not nested, then there exist two intersecting sets $A, B \in \mathcal{T}_i$ such that neither is contained in the other. Without loss of generality we may assume $x_{i,A} \geq x_{i,B}$. We will construct another feasible solution x' to LP1 as follows:

- $x'_{i,A\cup B} = x_{i,B}$
- $x'_{i,A} = x_{i,A} - x_{i,B}$
- $x'_{i,B} = 0$
- $x'_{i,A\cap B} = x_{i,B}$ if $A \cap B \neq \emptyset$
- $x'_{i,S} = x_{i,S}\ \forall\ S \subseteq X$ other than above.
- $x'_{j,S} = x_{j,S}\ \forall j \neq i\ ,\ \forall S \subseteq X$

By submodularity, one can verify that the cost of the solution x' is at most the cost of x. Let $\mathcal{T}'_i$ be the sets in the support of the new solution. Observe that $\sum_{S\in\mathcal{T}_i} |S|^2 < \sum_{S\in\mathcal{T}'_i} |S|^2$. Since $\sum_{S\in\mathcal{T}_i} |S|^2$ is monotonically increasing we can iterate the above process polynomially many times until we find a solution whose cost is no greater that that for x and the sets corresponding to its support form a nested family. □

Let $\overline{x}$ be an optimal solution of LP1 which satisfies the conditions in Lemma 62 and $\{\mathcal{T}_i \mid i \in [m]\}$ be the corresponding nested families of sets. Our algorithm is similar to the greedy algorithm for weighted set-cover where the candidate sets are chosen from $\mathcal{T} = \bigcup_i \mathcal{T}_i$. It proceeds in several rounds and in each round we allocate a subset of items to a single agent. Let Z_t be the set of unallocated items at the start of the t^{th} round. Define $\alpha_t(i, S) = f_i(S)/|S \cap Z_t|$. In each round we pick the set $(i, S) \in \mathcal{T}$ having the lowest value of $\alpha_t(i, S)$ (let α_t be this value) and assign the set S to agent i. We then update Z_t and proceed to the next round. Finally since the sets in $\mathcal{T}_i$ form a nested family each agent can simply pick the largest set assigned to her and drop all other sets.

Analysis: Let OPT be the cost of the optimal integral solution. Consider the t^{th} round of the algorithm and let H_t be the set of previously unallocated items that get assigned to an agent in this round. For the sake of analysis let us divide the incremental cost of the newly assigned items in this round equally among them i.e. each newly allocated item costs $\alpha_t = f_i(S)/|S \cap Z_t|$.

Next observe that α_t is at most $\mathrm{OPT}/|Z_t|$. This is because $\overline{x}$ is fractional solution that covers the items in Z_t and we picked the set (i, S) with the lowest value of

$f_i(S)/|S \cap Z_t|$. Thus the total cost of covering all items is at most

$$\sum_t \sum_{u \in H_t} \alpha_t \;\leq\; \sum_t \sum_{u \in H_t} \mathrm{OPT}/|Z_t| \;\leq\; \log n \cdot \mathrm{OPT}$$

The last inequality follows since $n = |Z_1| > |Z_2| > \cdots > |Z_t|$ and $\sum_{i \in [n]} 1/i = \log n$. To prove that this algorithm is also m-approximate, observe that each set selected has cost at most OPT. Moreover, each agent is assigned at most one set in the final solution. Thus we have the following theorem

Theorem 63. *There is a* $\min\{m, \log(n)\}$ *approximate algorithm for the Combinatorial Reverse Auction problem for agents with submodular cost functions.*

6.2.3 Perfect Matching

In this section, we consider the multi-agent submodular minimum perfect matching problem. In this problem we are given a bipartite graph $G(V, E)$ where $|V| = n$, containing at least one perfect matching and a normalized monotone submodular function $f_i : 2^E \longrightarrow R^+$ for each agent i. We wish to find a perfect matching M, and a partition of M into $M_1, M_2, \cdots, M_m$ such that $\sum_i f_i(M_i)$ is minimized. We first prove an information theoretic lower bound of $\Omega(n)$ on the approximability of the single agent case, which also implies the same bound for the multi-agent case. Then, we give an n-approximate algorithm for the multi-agent case.

6.2.3.1 ***Proof of Hardness***

As earlier, we proceed by designing two submodular functions that are hard to distinguish in polynomially many queries but have widely differing optimal values. In the general framework outlined in Section 6.2.1, this is accomplished by 'hiding' a random element of lower cost from the target collection C in one of the functions. In this case, C is the set of all perfect matchings. A natural first guess would be to pick a random perfect matching and hide the edges in it. This however does not serve our purpose because for a fixed pair of edges, the events that these edges belong to the

random matching are not independent, thus precluding the use of Chernoff bounds. We circumvent this problem by using the following result from the theory of random graphs [7]:

Theorem 64. *Let $G(n, n, p)$ be a random bipartite graph on $2n$ vertices such that each edge is present independently with probability p. Then*

$$Pr[\ G(n, n, p)\ \textit{contains no perfect matching}\ ] = O(ne^{-np})$$

Now instead of hiding a randomly chosen perfect matching, we hide a collection of randomly and independently chosen edges that contains a perfect matching with high probability. We prove the following theorem.

Theorem 65. *Any randomized approximation algorithm for the submodular minimum cost perfect matching problem with factor $O\left(\frac{n}{\log^2 n}\right)$ needs super-polynomially many queries.*

Proof. Consider the complete bipartite graph $K_{n,n}$. We choose a random subset R of edges by picking each edge independently with probability $p = \log^2 n/n$.

Define the following two submodular cost functions $f_R, g : 2^E \longrightarrow \mathbb{R}^+$:

$$\begin{aligned} f_R(Q) &= \min\{\ |Q \cap \overline{R}|\ +\ \min\{\ |Q \cap R|,\ (1+\delta)\log^2 n\ \},\ n\ \} \\ g(Q) &= \min\{\ |Q|,\ n\ \} \end{aligned}$$

Here δ is an arbitrary positive constant less than 1. Using Theorem 64, R contains a perfect matching with probability $1 - O(ne^{-\log^2 n})$, and hence the minimum cost of a perfect matching in f_R is at most $(1+\delta)\log^2 n$. Therefore the ratio of optima in g and f_R is $\Omega\left(\frac{n}{\log^2 n}\right)$ with high probability.

Now we look at the probability that the algorithm can not distinguish f_R and g. By Observation 5 it suffices to prove that $Pr[f_R(Q) \neq g(Q)]$ is super-polynomially small for an arbitrary query Q. It's easy to see that $f_R(S) \leq g(S)$, thus these two functions differ on Q if and only if $f_R(Q) < g(Q)$.

Let Q^* be the optimal query for which $Pr[\ f_R(Q) < g(Q)\ ]$ is maximized. We will show that $|Q^*| = n$. First, suppose that $|Q| \geq n$, then

$$\begin{aligned} Pr[\ f_R(Q) < g(Q)\ ] &= Pr[\ f_R(Q)\ <\ n\ ] \\ &= Pr\left[\ |Q \cap \overline{R}|\ +\ \min\left\{\ |Q \cap R|,\ (1+\delta)\log^2 n\ \right\}\ <\ n\ \right] \end{aligned}$$

which increases as the size of Q is reduced.

Now suppose $|Q| \leq n$. In this case,

$$\begin{aligned} Pr[\ f_R(Q) < g(Q)\ ] &= Pr[\ f_R(Q) < |Q|] \\ &= Pr\left[\ |Q \cap \overline{R}|\ +\ \min\left\{\ |Q \cap R|,\ (1+\delta)\log^2 n\ \right\}\ <\ |Q|\ \right] \\ &= Pr\left[\ \min\left\{\ |Q \cap R|,\ (1+\delta)\log^2 n\ \right\}\ <\ |Q \cap R|\ \right] \\ &= Pr[\ |Q \cap R|\ >\ (1+\delta)\log^2 n\ ] \end{aligned}$$

which increases as $|Q|$ is raised. Therefore, the optimal query size is n.

Therefore,

$$Pr[\ f_R(Q) < g(Q)\ ] = Pr[\ |Q \cap R| > (1+\delta)\log^2 n\ ]$$

Since $E[|Q \cap R|] = \log^2 n$ and edges were picked uniformly at random, we can apply Chernoff bounds to conclude that this probability is $O(e^{-\delta^2 \log^2 n})$. This proves the theorem. □

6.2.3.2 ***A n approximation algorithm***

Define a new cost function w over E as $w_e = \min_i f_i(\{e\})$ and define $w(Z) = \sum_{e \in Z} w_e$ for all $Z \subseteq E$. Since w is an additive valuation function we can find a minimum cost perfect matching in polynomial time. Let M be such a matching. Assign each edge $e \in M$ to the agent having the minimum valuation for that edge. Let the cost of this solution under the original valuation functions be W.

Analysis: We now prove that this is an n-approximate algorithm. By submodularity we have $W \leq w(M)$. Let M^* be an optimal solution of MS-MPM having value OPT. Since M is a minimum weight matching under w, $w(M) \leq w(M^*)$.

Let $w_{max} = \max_{e \in M_0}\{f_i(e) \mid \text{e assigned to agent } i \text{ in } M^*\}$. By submodularity of the cost functions, $w(M^*) \leq n \cdot w_{max}$. By monotonicity we have $w_{max} \leq OPT$. Therefore,

$$W \leq w(M) \leq w(M^*) \leq n \cdot w_{max} \leq n \cdot OPT$$

This yields the following theorem

Theorem 66. *There is a n approximate algorithm for the Perfect Matching problem for agents with submodular cost functions.*

6.3 Discounted Cost Functions

In this section we will study the approximability of allocation problems under discounted cost functions. Recall that a discounted cost function is given by an additive cost function $c : X \to R^+$ and a discount function $d : R^+ \to R^+$ that satisfies the following properties: (1) $d(0) = 0$; (2) d is increasing; (3) $d(x) \leq x$ for all x; (4) d is concave. The price of any $S \subseteq X$ is defined to be $d(c(S))$.

6.3.1 Combinatorial Reverse Auction

Since discounted cost functions are a special case of submodular cost functions, the result from Theorem 63 yields the following theorem.

Theorem 67. *There is a* $\min\{m, \log(n)\}$ *approximate algorithm for the Combinatorial Reverse Auction problem for agents with discounted cost functions.*

Next we show a lower bound this problem that is contingent upon $P \neq NP$ and that follows from a simple reduction from the hardness of Set Cover.

Theorem 68. *It is hard to approximate the discounted reverse auction problem within factor* $(1 - o(1)) \log n$ *unless* $\mathrm{P} = \mathrm{NP}$.

Proof. We reduce set cover to the discounted reverse auction problem to prove this result. Consider an instance $\mathcal{I} = (U, C, w)$ of set cover where we wish to cover all

elements in the universe U using sets from C and minimize the sum of weights under the weight function $w : C \to \mathrm{R}^+$. We define an instance, $\mathcal{I}'$ of our discounted reverse auction problem corresponding to $\mathcal{I}$ in the following way. Let U be the set of items. For every set $S \in C$ define an agent a_S, whose cost function c_a assigns the value $w(S)$ for every element $s \in S$ and sets the cost of all other elements in U to be infinity. The discounted price function for the agent is shown in figure 8. Here the slope of the second segment is small enough.

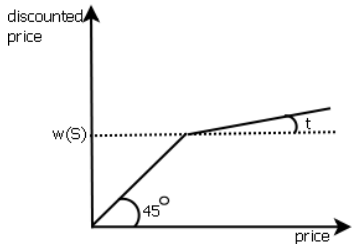

Figure 8: Discount function for agent corresponding to set S

Consider a solution for $\mathcal{I}'$ where we procure at least one item from agent a_S; then we can buy all elements in S from a_S without a significant increase in our payment. So the cost of the optimal solution to $\mathcal{I}$ can be as close to the price of the optimal solution for $\mathcal{I}'$ as we want. By [19], set cover is hard to approximate to a factor better than $\log n$ unless P = NP. Therefore the discounted reverse auction problem can not be approximated within factor $(1 - o(1)) \log n$ unless P = NP. □

6.3.2 Perfect Matching

Next we consider the Perfect Matching problem under discounted cost functions. Once again the trivial algorithm from Theorem 66 carries over and gives the following theorem.

Theorem 69. *There is a n approximate algorithm for the Perfect Matching problem for agents with discounted cost functions.*

As for the lower bound, Theorem 68 immediately gives an $O(\log(n)$ lower bound

for this problem. In Theorem 73 we will amplify this bound to show a polylog(n) lower bound for this problem. Our amplification technique is somewhat indirect - We first show a polylog lower bound on the shortest path problem under discounted cost functions and then show a reduction to the perfect matching problem to complete the proof.

In the $s-t$ path problem with discounted costs we are given a connected graph $G(V,E)$ and two chosen vertices $s,t \in V$. There are m agents each having a discounted cost function over E who wish to collectively build a minimum cost path from s to t. In Theorem 70 we show that unless P = NP, this problem is hard to approximate to within $O(\log^c n)$ for any fixed constant c.

Theorem 70. *The discounted shortest $s-t$ path problem is hard to approximate within a factor of $O(\log^c n)$ for any fixed constant $c > 0$.*

Proof. Our proof proceeds by repeatedly applying a transformation σ on the given family of problem instances, which amplifies the approximation factor on every application. Each application of σ also increases the size of the graph but only by a polynomial(in n) factor. We now describe the transformation formally.

Consider the following instance $(G,s,t,\mathcal{A},\mathcal{U})$: We are given a graph $G=(V,E)$, vertices s,t and a set $\mathcal{A}$ of agents. We are also given a collection $\mathcal{U}=\{U_a\}_{a\in\mathcal{A}}$. Here $U_a \subseteq E$ specifies the set of edges that can be assigned to agent a, i.e., $c_a(e)=1$ for all $e \in U_a$ and $c_a(e)=+\infty$ otherwise. The discounted price function d_a is such that $d_a(x)=x$ for all $x \leq 1$ and 1 for all $1 < x < +\infty$. Observe that under this assumption, for any set S of edges, $d_a(S)$ has value 1 if $S \subseteq U_a$ and ∞ otherwise. We may assume that the sets U_a for $a \in \mathcal{A}$ are pairwise disjoint, by replacing a single edge that can be assigned to multiple agents by parallel edges and assigning them to each of the agents. In future discussion, we will use $\mathcal{F}$ to denote the family of instances $\{(G,s,t,\mathcal{A},\mathcal{U})\}$.

We define the transformation $\sigma : \mathcal{F} \to \mathcal{F}$ that takes an instance $I=(G,\mathcal{A},\mathcal{U})$

in $\mathcal{F}$, and generates another instance $I^{\otimes} = (G^{\otimes}, \mathcal{A} \times \mathcal{A}, \mathcal{U}^{\otimes})$ as follows. The graph $G^{\otimes} = (V^{\otimes}, E^{\otimes})$ is constructed from G by replacing each edge $(u, v) \in E$ with a copy of the graph G such that s coincides with u and t coincides with v. Thus any edge $e \in E$ can be identified with a subgraph G^e, of $G^{\otimes}$ that is isomorphic to G. For any $e' \in G^e$ define $\rho(e') = e$ and define $\gamma(e')$ to be the edge corresponding to e' in G under this isomorphism. There are $|\mathcal{A}|^2$ agents in the new instance who are indexed by $\mathcal{A} \times \mathcal{A}$. We define the elements of $\mathcal{U}^{\otimes}$ as $U^{\otimes}_{(a_1,a_2)} = \{e' \mid \rho(e') \in U_{a_1} \text{ and } \gamma(e') \in U_{a_2}\}$.

Note that $|E^{\otimes}| = |E|^2$, i.e. the size of instance $I^{\otimes}$ is bounded by a polynomial in the size of I. We define $\sigma(\mathcal{F}) = \{\sigma(I) \mid \forall I \in \mathcal{F}\}$. In lemma 71, we show that we can amplify that hardness result from theorem 68 by applying the transformation σ repeatedly.

Lemma 71. *If $\mathcal{H} = \sigma^r(\mathcal{F})$ is a family of instances for the $s-t$ path problem that is hard to approximate to a factor better than α then $\sigma(\mathcal{H})$ is hard to approximate within factor $O(\alpha^2)$.*

Proof. Let $I = (G, s, t, \mathcal{A}, \mathcal{U})$ be an instance in $\mathcal{H}$. Let us begin by making some observations about the structure of an optimal solution for $\sigma(I) = (G^{\otimes}, s, t, \mathcal{A}, \mathcal{U}^{\otimes})$.

Claim 16. *If there is a $s-t$ path of price β in G then there is a $s-t$ path in $G^{\otimes}$ of price at most β^2.*

Proof. Let $P = e_1, e_2 \ldots e_t$ be a path of price β in G. We can construct a $s-t$ path in $G^{\otimes}$ by considering the set of graphs $G^{e_1} \ldots G^{e_t}$ and picking the edges corresponding to the edges in P in each of these copies. It can be verified that this gives us a valid path that has price β^2. □

Next we note that the converse is also true.

Claim 17. *If there is a $s-t$ path of price β^2 in G' then there is a $s-t$ path in G of price at most β.*

Proof. Let P be a path of price β^2 in $G^{\otimes}$. Let $G^{e_1} \dots G^{e_t}$ be the copies of G that have non-empty intersection with P. Two cases may arise. Either the set of edges $\{e_1 \dots e_t\}$ belong to at most β distinct agents in $\mathcal{A}$ or they belong to more than β agents in $\mathcal{A}$. Note that the set of edges $\{e_1 \dots e_t\}$ form a path in G, and in the first case this path has price at most β. In the second case, the price of edges in $P \cap G^{e_i}$ must be less than β for some copy G^{e_i} of graph G. These edges also form a $s-t$ path in G of price at most β. Thus in both cases we can find a path of price at most β in G. □

Using the observations above, if the price of the optimal solution to I is OPT, then the price of the optimal solution to $\sigma(I)$ is OPT^2. Furthermore, if we can approximate the optimal solution to $\sigma(I)$ to within a factor of $o(\alpha^2)$ then we can approximate the optimal solution for I to better than $o(\alpha)$, using the construction in claim 17. This yields the desired contradiction. □

Since $\mathcal{F}$ is hard to approximate to within a factor of $\log n$ applying lemma 71 repeatedly gives the desired result. □

Finally we show that the result of Theorem 70 also carries over to the perfect matching problem by a reduction from the $s-t$ path problem.

Lemma 72. *Let A be a β-approximate algorithm for the perfect matching problem, then we can get a β-approximation for the $s-t$ path problem using A as a subroutine.*

Proof. Suppose we are given a graph $G = (V, E)$. Construct an auxiliary graph G^* in the following way: Replace every vertex $v \in V$ by v' and v'' and add an edge connecting them. The price of this edge is zero for every agent. We replace each edge $uv \in E$ with the gadgets shown in figure 9.

On this graph G^*, use the algorithm A to get the minimum weight matching. Let M be the matching returned. We can interpret M as a $s-t$ path in G in the following

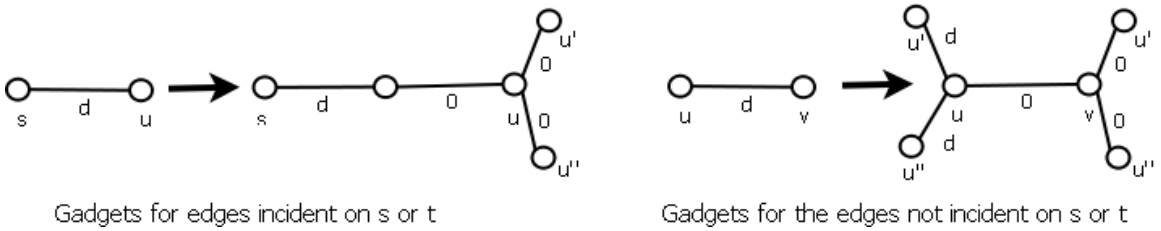

Figure 9: Gadgets

way. Let $g(uv)$ be the edges in G^* corresponding to the edge uv for the gadget shown in figure 9. Observe that either one or two edges of every such gadget must belong to M. Let S be the set of edges in G such that two edges in their corresponding gadget belong to M. One can check that every vertex in V is incident with zero or two edges from S, whereas s and t are each incident with exactly one edge in S. Therefore S consists of an $s-t$ path P_S and some other circuits. Now the circuits in S must have cost zero. This is because if a circuit has positive cost then the cost of the matching can be reduced further by pairing up the vertices in the circuit as shown in figure 10.

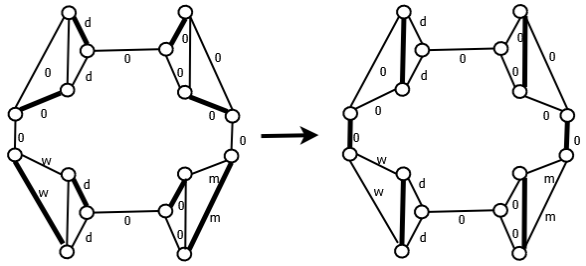

Figure 10: Circuits not involving edges in S should have zero costs

□

Note that the reduction defined in lemma 72 defines a cost preserving bijection between $s-t$ paths in G to perfect matchings in G^*. Therefore, using Theorem 70 we have the following theorem.

Theorem 73. *The discounted perfect matching problem is hard to approximate within a factor of* $O(\log^c n)$ *for any fixed constant* $c > 0$.

6.4 Conclusion and Open Problems

In this chapter we initiated a systematic study of allocation problems under non-linear cost functions. We investigated the approximability of two classical allocation problems - combinatorial reverse auctions and minimum cost perfect matching - and showed upper and lower bounds for both of them.

The setting that we have considered in this work is quite general, and is a very exciting avenue of research. There are many other interesting problems in this class such as minimum graph cut and edge cover which could be studied in the future work. We have considered the covering problems in this work, one can ask the same questions for packing problems. One could also consider even more general functions like subadditive functions. Extension to multi-agent systems makes a natural connection to Game Theory. Mechanism design of these combinatorial problem also has interesting applications.

It is surprising to note that CRA which is NP-hard even for linear cost functions does not get much harder for submodular or discounted cost functions. On the other hand the minimum perfect matching problem can be solved efficiently for linear cost functions, but it turns out to be extremely hard to approximate for non-linear functions. It would be interesting to come up with a characterization of problems whose computational complexity remains unchanged for both linear and non-linear cost functions.

REFERENCES

[1] Aggarwal, G., Goel, G., Karande, C., and Mehta, A., "Online vertex-weighted bipartite matching and single-bid budgeted allocations," *SODA*, 2010.

[2] Aronson, J., Dyer, M., Frieze, A., and Suen, S., "Randomized greedy matching. ii," *Random Struct. Algorithms*, vol. 6, pp. 55–73, January 1995.

[3] Aronson, J., Dyer, M., Frieze, A., and Suen, S., "Randomized greedy matching. ii," *Random Struct. Algorithms*, vol. 6, pp. 55–73, January 1995.

[4] Bahmani, B. and Kapralov, M., "Improved bounds for online stochastic matching," *ESA*, 2010.

[5] Bansal, N., Gupta, A., Li, J., Mestre, J., Nagarajan, V., and Rudra, A., "When lp is the cure for your matching woes: improved bounds for stochastic matchings," in *Proceedings of the 18th annual European conference on Algorithms: Part II*, ESA'10, pp. 218–229, 2010.

[6] Birnbaum, B. and Mathieu, C., "On-line bipartite matching made simple," *SIGACT News*, 2008.

[7] Bollobas, B., *Random Graphs*. Cambridge University Press, 2001.

[8] Buchbinder, N., Jain, K., and Naor, J., "Online Primal-Dual Algorithms for Maximizing Ad-Auctions Revenue," in *Algorithms–ESA 2007 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007: Proceedings*, p. 253, Springer, 2007.

[9] Calinescu, G., Chekuri, C., Pál, M., and Vondrák, J., "Maximizing a submodular set function subject to a matroid constraint (extended abstract),"

in *IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization*, pp. 182–196, 2007.

[10] Chakrabarty, D. and Goel, G., "On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap," *SIAM J. Comput.*, vol. 39, pp. 2189–2211, March 2010.

[11] Chen, N., Immorlica, N., Karlin, A. R., Mahdian, M., and Rudra, A., "Approximating matches made in heaven," in *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pp. 266–278, 2009.

[12] Costello, K., Tetali, P., and Tripathi, P., "Stochastic matching with commitment," ICALP, 2012.

[13] Dean, B. C., Goemans, M. X., and Vondrák, J., "Adaptivity and approximation for stochastic packing problems," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, (Philadelphia, PA, USA), pp. 395–404, Society for Industrial and Applied Mathematics, 2005.

[14] Devanur, N. and Hayes, T., "The adwords problem: Online keyword matching with budgeted bidders under random permutations," in *ACM Conference on Electronic Commerce*, 2009.

[15] Duan, R. and Pettie, S., "Approximating maximum weight matching in near-linear time," in *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, (Washington, DC, USA), pp. 673–682, IEEE Computer Society, 2010.

[16] Dyer, M. and Frieze, A., "Randomized greedy matching," *Random Structures and Algorithms*, vol. 2, no. 1, pp. 29–45, 1991.

[17] Edmonds, J., "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.

[18] Farkas, J. G., "Uber die theorie der einfachen ungleichungen," *Journal fur die Reine und Angewandte Mathematik*, vol. 124, pp. 1–27, 1902.

[19] Feige, U., "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[20] Feige, U., Mirrokni, V. S., and Vondrak, J., "Maximizing non-monotone submodular functions," in *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pp. 461–471, 2007.

[21] Feige, U. and Vondrak, J., "Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e," in *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pp. 667–676, 2006.

[22] Feldman, J., Mehta, A., Mirrokni, V. S., and Muthukrishnan, S., "Online stochastic matching: Beating 1-1/e," in *FOCS*, pp. 117–126, 2009.

[23] Goel, G., Karande, C., Tripathi, P., and Wang, L., "Approximability of combinatorial problems with multi-agent submodular cost functions," *Foundations of Computer Science, Annual IEEE Symposium on*, pp. 755–764, 2009.

[24] Goel, G. and Mehta, A., "Online budgeted matching in random input models with applications to adwords," in *SODA*, pp. 982–991, 2008.

[25] Goel, G. and Tripathi, P., "Matching with your eyes closed," *Foundations of Computer Science, Annual IEEE Symposium on*, 2012.

[26] Goel, G., Tripathi, P., and Wang, L., "Combinatorial Problems with Discounted Price Functions in Multi-agent Systems," in *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, vol. 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 436–446, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[27] Goemans, M. X., Harvey, N. J. A., Iwata, S., and Mirrokni, V., "Approximating submodular functions everywhere," in *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 535–544, 2009.

[28] Hayrapetyan, A., Swamy, C., and Tardos, E., "Network design for information networks," in *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 933–942, 2005.

[29] Hoeffding, W., "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, pp. 13–30, March 1963.

[30] Iwata, S. and Nagano, K., "Submodular function minimization under covering constraints," in *Foundations of Computer Science, 2009. FOCS '09. 50th Annual IEEE Symposium on*, pp. 671 –680, Oct. 2009.

[31] Iwata, S., "A faster scaling algorithm for minimizing submodular functions," *SIAM Journal of Computing*, vol. 32, no. 4, pp. 833–840, 2003.

[32] Iwata, S., "Submodular function minimization," *Mathematical Programming*, vol. 112, no. 1, pp. 45–64, 2008.

[33] Iwata, S., Fleischer, L., and Fujishige, S., "A combinatorial strongly polynomial algorithm for minimizing submodular functions," *J. ACM*, vol. 48, no. 4, pp. 761–777, 2001.

[34] Iwata, S. and Orlin, J. B., "A simple combinatorial algorithm for submodular function minimization," in *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pp. 1230–1237, 2009.

[35] Jain, K., Mahdian, M., and Saberi, A., "A new greedy approach for facility location problems," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, (New York, NY, USA), pp. 731–740, ACM, 2002.

[36] Karande, C., Mehta, A., and Tripathi, P., "Online bipartite matching in the unknown distributional model," in *STOC*, pp. 106–117, 2011.

[37] Karp, R., Vazirani, U., and Vazirani, V., "An optimal algorithm for online bipartite matching," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990.

[38] Khot, S., Lipton, R. J., Markakis, E., and Mehta, A., "Inapproximability results for combinatorial auctions with submodular utility functions," *Algorithmica*, vol. 52, no. 1, pp. 3–18, 2008.

[39] Korula, N. and Pál, M., "Algorithms for secretary problems on graphs and hypergraphs," in *Proceedings of the 36th Internatilonal Collogquium on Automata, Languages and Programming: Part II*, ICALP '09, pp. 508–520, 2009.

[40] Kurtz, T. G., "Solutions of ordinary differential equations as limits of pure jump markovprocesses.," *Journal of Applied Probability*, vol. 7, pp. 49–58, 1970.

[41] Lehman, B., Lehman, D., and Nisan, N., "Combinatorial auctions with decreasing marginal utilities," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, pp. 18 –28, 2001.

[42] Mahdian, M., Nazerzadeh, H., and Saberi, A., "Allocating online advertisement space with unreliable estimates," in *Proceedings of the 8th ACM conference on Electronic commerce*, EC '07, (New York, NY, USA), pp. 288–294, ACM, 2007.

[43] Mahdian, M., Nazerzadeh, H., and Saberi, A., "Allocating online advertisement space with unreliable estimates," in *ACM Conference on Electronic Commerce*, pp. 288–294, 2007.

[44] Mahdian, M. and Yan, Q., "Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps," in *STOC*, pp. 117–126, 2011.

[45] Mahdian, M., Ye, Y., and Zhang, J., "Improved approximation algorithms for metric facility location problems," in *In Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pp. 229–242, 2002.

[46] Manshadi, V. H., Oveis-Gharan, S., and Saberi, A., "Online stochastic matching: Online actions based on offline statistics," in *SODA*, 2011.

[47] Mehta, A., Saberi, A., Vazirani, U., and Vazirani, V., "Adwords and generalized online matching," in *FOCS*, 2005.

[48] Micali, S. and Vazirani, V. V., "An o(sqrtve) algoithm for finding maximum matching in general graphs," in *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pp. 17 –27, oct. 1980.

[49] Mirrokni, V. S., Schapira, M., and Vondrák, J., "Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions," in *ACM Conference on Electronic Commerce*, pp. 70–77, 2008.

[50] Nikolova, E., Kelner, J. A., Brand, M., and Mitzenmacher, M., "Stochastic shortest paths via quasi-convex maximization," in *Proceedings of the 14th conference on Annual European Symposium - Volume 14*, (London, UK), pp. 552–563, Springer-Verlag, 2006.

[51] Orlin, J. B., "A faster strongly polynomial time algorithm for submodular function minimization," in *IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization*, pp. 240–251, 2007.

[52] Papadimitriou, C. H. and Yannakakis, M., "Shortest paths without a map," *Theor. Comput. Sci.*, vol. 84, pp. 127–150, July 1991.

[53] Pettie, S. and Sanders, P., "A simpler linear time $2/3 - \epsilon$ approximation to maximum weight matching," *Information Processing Letters*, vol. 91, no. 6, pp. 271–276, 2004.

[54] Ross, L., Rubin, D., Siegler, M., Josephson, M., Thistlethwaite, J., and E.S.Woodle, "Ethics of a paired-kidney-exchange program," *The New England Journal of Medicine*, vol. 336, pp. 1752–1755, 1997.

[55] Ross, L., Rubin, D., Siegler, M., Josephson, M., Thistlethwaite, J., and Woodle, E., "The case for a living emotionally related international kidney donor exchange registry," *Transplantation Proceedings*, vol. 18, pp. 5–9, 1986.

[56] Roth, A. E., Sonmez, T., and Unver, M. U., "A kidney exchange clearinghouse in new england," *American Economic Review, Papers and Proceedings*, vol. 95, pp. 376–380, May 2005.

[57] Roth, A. E., Sonmez, T., and Unver, M. U., "Pairwise kidney exchange," *Journal of Economic Theory*, vol. 125, pp. 151–188, December 2005.

[58] Roth, A. E., Sonmez, T., and Unver, M. U., "Kidney paired donation with compatible pairs(letter to the editor)," *American Journal of Transplantation*, vol. 7, December 2007.

[59] Schrijver, A., "A combinatorial algorithm minimizing submodular functions in strongly polynomial time," *Journal of Combinatorial Theory, Series B*, vol. 80, pp. 346–355, 2000.

[60] Sharma, Y., Swamy, C., and Williamson, D. P., "Approximation algorithms for prize collecting forest problems with submodular penalty functions," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1275–1284, 2007.

[61] Shmoys, D. B. and Swamy, C., "An approximation scheme for stochastic linear programming and its application to stochastic integer programs," *J. ACM*, vol. 53, pp. 978–1012, November 2006.

[62] Sviridenko, M., "A note on maximizing a submodular set function subject to a knapsack constraint," *Oper. Res. Lett.*, vol. 32, no. 1, pp. 41–43, 2004.

[63] Svitkina, Z. and Fleischer, L., "Submodular approximation: Sampling-based algorithms and lower bounds," in *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 697–706, 2008.

[64] Svitkina, Z. and Tardos, E., "Facility location with hierarchical facility costs," in *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pp. 153–161, 2006.

[65] Tao, T. and Vu, V., *Additive Combinatorics*. Cambridge University Press, 2006.

[66] Vinkemeier, D. E. D. and Hougardy, S., "A linear-time approximation algorithm for weighted matchings in graphs," *ACM Trans. Algorithms*, vol. 1, pp. 107–122, July 2005.

[67] Vondrák, J., "Optimal approximation for the submodular welfare problem in the value oracle model," in *STOC*, pp. 67–74, 2008.

[68] Wolsey, L. A., "An analysis of the greedy algorithm for the submodular set covering problem," *Combinatorica*, vol. 2, no. 4, pp. 385–393, 1982.

[69] Yao, A. C., "Probabilistic computations: towards a unified measure of complexity," *FOCS*, pp. 222–227, 1977.

VITA

Education

Georgia Institute of Technology, Atlanta, Georgia

Algorithms, Combinatorics and Optimization. May 2012.

Indian Institute of Technology, Delhi, India

Bachelor of Technology, Computer Science and Engineering, August 2008.

Publications

1. Gagan Goel and Pushkar Tripathi. Matching with our Eyes Closed. *IEEE Symposium on Foundations of Computer Science (FOCS), 2012*

2. Chinmay Karande, Aranyak Mehta, Pushkar Tripathi. Online Bipartite Matching in the Unknown Distributional Model. *ACM Symposium on Theory of Computing (STOC), 2011*

3. Gagan Goel, Chinmay Karande, Pushkar Tripathi and Lei Wang. Approximability of Combinatorial Problems with Multi-agent Submodular Cost Functions. *IEEE Symposium on Foundations of Computer Science (FOCS), 2009*

4. Kevin Costello, Prasad Tetali and Pushkar Tripathi. Matching with Commitment. *International Colloquium on Automata, Languages and Programming (ICALP), 2012*

5. Satoru Iwata, Prasad Tetali and Pushkar Tripathi. Approximating Minimum Linear Ordering Problems. *APPROX , 2012*

6. Gagan Goel, Pushkar Tripathi and Lei Wang. Optimal Approximation Algorithms for Multi-agent Combinatorial Problems with Discounted Price Functions. *Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2010*

7. Pushkar Tripathi, Rohan Jain, Srikanth Kurra and P.R. Panda. REWIRED: Register Write Inhibition by Resource Dedication, *Asia Pacific Design Automation Conference (ASPDAC), 2008*

8. Ravish Mehra, Pushkar Tripathi, Niloy Mitra, Alla Sheffer. Visibility of Noisy Point Cloud Data. *Shape Modelling International, 2010.* Invited to special issue of Computers & Graphics.