Online Bipartite Matching under Markov Chain Model

Arnav Ishaan

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

December 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Arnav Ishaan**

Entitled: **Online Bipartite Matching under Markov Chain Model**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Hovhannes Harutyunyan

_____ Examiner
Dr. Hovhannes Harutyunyan

_____ Examiner
Dr. Lata Narayanan

_____ Supervisor
Dr. Denis Pankratov

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2024

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Online Bipartite Matching under Markov Chain Model

Arnav Ishaan

Online bipartite matching (OBM) has a rich history in the literature of online algorithms, where it has been an influential problem inspiring many algorithms and techniques. This problem of obtaining a matching set of maximum size where vertices of one bi-partition arrive online has many real-world applications from kidney donor exchange to online advertising. This has led to the problem being studied under a variety of input models. In the adversarial model it is known that the tight competitive ratio is $1 - 1/e$ (among all randomized algorithms). Lower and upper bounds on competitive ratios better than $1 - 1/e$ are known for random order model, known and unknown IID models and other stochastic models, where a major open problem is to close these gaps.

One feature of the stochastic input models (e.g., known and unknown IID input models) under which OBM has been studied so far is the assumption of strong independence among the input items. One of the main conceptual contributions of this thesis is to introduce a stochastic input model that allows us to simulate limited dependence. In our model, input nodes are sampled from a Markov chain, and we refer to this as the Markov chain model. Introducing Markov chain significantly increases the complexity of analysis of algorithms by adding a number of parameters: initial distribution, transition probabilities, sampling size, etc, which leads us to concentrate on analyzing the problem for some specific families of bipartite graphs and Markov chains.

In particular, we study two algorithms Non Adaptive and Adaptive Two Suggested Matching under the Markov chain input model for parameterized versions of lazy random walks on $(2, 2)$-biregular type graphs. We give an alternative characterization of an offline optimal solution, OPT, for these stochastic inputs, which allows us to calculate exactly (in the limit) the expected size of

matching of OPT. We then proceed to obtain tight bounds on the sizes of matchings obtained by the two algorithms under asymptotic conditions, which combined with the bound on OPT, gives us tight competitive ratios of 0.9509 and 0.9733 for the Non Adaptive and Adaptive algorithms, respectively. These are competitive ratios for the classical lazy random walk Markov chain, where the probability of staying put is $1/2$. We also derive exact formulas for competitive ratios with respect to lazy walks parameterized by p – the probability of staying put.

We believe these results, which use two disjoint matchings and regularity of graph degrees could be extended to type graphs of degree at most k (for any constant k), and to bipartite graphs that admit several disjoint matchings of large sizes.

Acknowledgments

I would like to thank my supervisor, Dr. Denis Pankratov for his unwavering support and guidance throughout my graduate studies. Two years ago, I sought his mentorship after being fascinated by his teaching methodology and approach to problem solving in the Design and Analysis of Algorithms class. He generously took me under his tutelage, and since then, he has patiently guided me through my graduate journey, helping me hone my skills and research temperament. I am truly grateful for his advice, feedback, and the rapport we share. I have thoroughly enjoyed working under his expertise, and I hope to replicate an acumen similar to his in both my professional and personal life in the future.

The past two years in Montreal have been enriched by the amazing professors and lab mates in the Algorithms and Complexity Group at Concordia University, who have made my journey exciting and fulfilling. I am deeply appreciative of the time and words I have shared with this wonderful group. I would like to express my heartfelt thanks to them, as well as to all the faculty and staff members of Concordia University.

On a personal note, I would like to thank my friends, peers, and well-wishers for their camaraderie and encouragement at different stages of my life.

Lastly, none of this would have been possible without the unwavering love, support, and sacrifices of my parents and younger brother, to whom I am forever indebted.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a sponsored search scenario, a user submits a query to a search engine, and the search engine returns a list of results alongside a couple of ads that are deemed relevant to the search query. The ad matching platform has a set of advertisers and knows about the types of users the advertisers are interested in. The platform wants to maximize the number of relevant ads shown to the 'interested' users, or *impressions* in the regime of ad matching. Note that the impressions arrive sequentially, and the platform needs to decide on which ad to show without knowing future impressions. This is known as an *online* problem.

More generally, consider a problem in which input is represented by a sequence of input items. An algorithm is required to make a decision for each input item with the goal of optimizing an objective function, which evaluates how good the decisions are for the given input. In an offline scenario, each decision can depend on the entire sequence of input items. In an online scenario, i^{th} decision can depend only on input items $1, 2, \ldots, i$. That is, in an online setting the algorithm is required to make an irrevocable decision without seeing future input items. Performance of an online algorithm is measured by *competitive ratio*, which is the worst-case ratio (over input instances) between the value obtained by the online algorithm and the value obtained by an optimal offline algorithm.

The underlying combinatorial problem behind the sponsored search scenario is online bipartite matching. Given a bipartite graph $G(U, V, E)$, with vertex set U known in advance, and vertices with type (or being similar to vertices) in V arriving in an online fashion, an algorithm needs to

construct a *matching* – an independent edge set, where no two edges share a common vertex, of largest size possible.

The problem of online bipartite matching was introduced in the seminal work of Karp, Vazirani, Vazirani in 1990 [51], which presented a randomized online algorithm with competitive ratio $1 - \frac{1}{e}$ in adversarial setting. They also showed that this bound is tight, which seemed to settle the problem of online bipartite matching at least for a while. With the boom of the internet and the explosion in online advertisement, there was a renewed interest in this problem beginning in the late 2000s, and it was studied again under a different input model.

The adversarial assumption that the algorithm has absolutely no information about future input is often too restrictive. In practice, oftentimes extra information is available to an algorithm, particularly, in the form of historical statistical data. This naturally leads to the following two questions: (1) can we use extra information to design better matching algorithms? and (2) can we use extra information to give a better analysis of the performance of existing algorithms? In particular, in sponsored search, the platform can collect how often different types of users are accessing the system. This naturally leads to an input model known as the IID model. The IID stands for independent identically distributed input items. Motivated by these considerations, Feldman et al. [26] studied online bipartite matching under the IID setting and presented an algorithm which achieved a competitive ratio of 0.67 (with high probability) breaking the $1 - \frac{1}{e}$ barrier for the first time in nearly 20 years. Since then, it has been followed up by many improvements in the IID setting [6, 62, 45].

As evidenced by the literature cited above, up until now most of the works on online bipartite matching considered statistical information with strong independence assumptions. This thesis is the first work in the matching area that considers statistical information with limited dependence, motivated by the following scenario. Consider the sponsored search again and suppose there is a major event happening, such as the World Cup finals. It is reasonable to assume that a lot of consecutive search queries received by the search platform are not independent, but have a higher chance of being related to a major event (such as looking up player statistics for the soccer players involved in a game). This corresponds to many searches being done by users who are soccer enthusiasts, i.e., of a similar type.

We model the above scenario as follows. There is a bipartite *type* graph known to the algorithm in advance. One side of the graph represents types of users, while the other side of the graph represents different advertisers. There is an edge between a user type and an advertiser, if they are potentially a good match. There is also a *Markov chain* with the state space being the set of user types. A sequence of users of different types is drawn from the Markov chain, and they need to be matched to advertisers, so as to maximize the total size of the matching at the end. Having a Markov chain allows one to model that a user of a particular type is more or less likely to follow the previous user of a particular type. The main conceptual contribution of the thesis is to introduce this new input model. We note that adding a Markov chain to the picture substantially increases the complexity of the model. Markov chain itself adds a quadratic (in the number of user types) number of parameters to the model in addition to the initial distribution, number of samples, and the type graph parameters. Thus, the Markov chain input model is rather general. In particular, it generalizes IID models and even can be used to model adversarial setting (albeit, under a somewhat unnatural setting of parameters). This necessitates the restriction of parameters to some special classes. We demonstrate that it is possible to analyze algorithms under this model and even obtain tight competitive ratios, which constitute the main technical contributions of this thesis.

More specifically, we consider the online matching problem in $(2, 2)$-biregular bipartite type graphs under a family of Markov chains, representing parameterized lazy random walks. We analyze two versions of a previously introduced online algorithm, Two Suggested Matching. We calculate the expected size of matching in the online setting and present tight bounds on the asymptotic competitive ratios of these two algorithms. We also motivate studying the setting of $(2, 2)$-biregular bipartite type graphs and parameterized lazy random walks by examining real-world data related to sponsored ads.

This thesis is structured the following way – we begin by introducing some notations, mathematical definitions, and problem models in the preliminaries chapter. Next, in Chapter 3, we provide a brief background on the work done on online bipartite matching under various input models. In Chapter 4, we give an alternative characterization of the optimal solution in $(2, 2)$-biregular bipartite graphs under the family of Markov chains stated above. This alternative characterization then allows us to derive an asymptotically tight closed-form formula for the expected size of the matching in

the offline setting. In Chapter 5, we analyze two online algorithms and calculate the expected size of online matching. Combining this with the results of Chapter 4, we obtain tight bounds on the competitive ratios of the two algorithms in the considered regimes. In Chapter 6, we explain the experiment by which we extract a bipartite type graph and Markov chain based on statistics from a real world example of online advertisement. Finally, we summarize the results and present some open questions in Chapter 7.

Chapter 2

Preliminaries

In this section, we provide a brief background on the mathematical structures which we use in our thesis. We present some definitions and introduce notations to maintain consistency throughout the results. We start by providing background on graphs and matching – the underlying structure and problem of our works. We then provide some brief background on online algorithms, and present the problem of online bipartite matching. As discussed in the previous chapter, online matching has been studied under different models, which we summarize here, and present a new variation of the model online bipartite matching under Markovian inputs.

2.1 Graph Theory

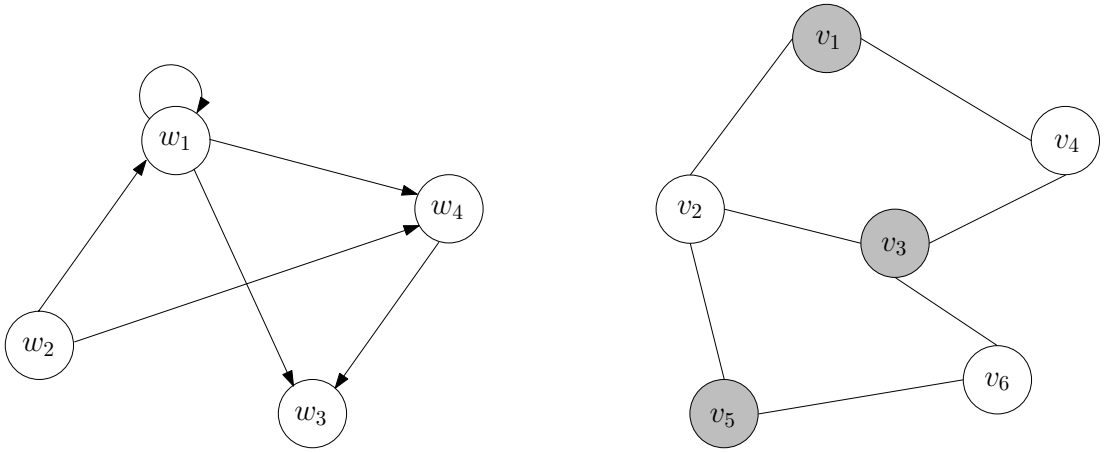
2.1.1 Graphs

In discrete mathematics and computer science, graphs are a way of representing a set of entities which are related to each other. The objects are known as *vertices* or *nodes* and the relationship between them is denoted using *edges*. Mathematically, a *graph* $G = (V, E)$ is a pair, where V represents the set of vertices and E represents the set of edges. Directed graphs or *digraphs* are graphs whose edges are directed. An edge $e \in E$ of a directed graph is represented by a pair of vertices (v_1, v_2) where $v_1, v_2 \in V$. By abuse of terminology, we can interpret undirected graphs as special digraphs such that $(v_1, v_2) \in E$ if and only if $(v_2, v_1) \in E$. Weighted graphs are graphs in which a numerical value is assigned to the edges of the graph, which is used to represent some

metric, like distance, or cost associated with moving along the weight.

In this thesis, we focus on *simple* graphs - graphs with no parallel edges, which implies for a pair of vertex v_1 and v_2, there can be at-most 1 edge from v_1 to v_2. In case of undirected graphs, we also restrict the presence of self loops, which means the endpoints of an edge can't be the same vertex. For directed graphs, we allow the possibility of self loops[1]. Unless stated otherwise, from here on, graphs will be used to refer to simple graphs.

The *degree* of a vertex for a graph, denoted by $deg(v)$ is the total number of edges whose one of the end points are incident on the said vertex. In other words, the total number of vertices a vertex is connected to is the degree of the vertex. The *neighborhood* of a vertex v, is defined as the set of vertices which share an edge with v and is denoted by $N(v)$. Directed graphs have out-degree and in-degree which denote the outgoing and incoming edges to the vertex. We denote the in-degree and out-degree of vertex v as $deg^+(v)$ and $deg^-(v)$ respectively. A graph in which all vertices have the same degree is known as a *regular* graph.



(a) Example of a directed graph with self loop at vertex w_1

(b) Example of a bipartite graph, white and gray color denotes the two bipartite sets

Figure 2.1: Example of graphs.

A graph is said to be *bipartite* if the vertex set V can be bifurcated into 2 non empty, disjoint and independent subsets V_1 and V_2, such that every edge is between V_1 and V_2. In other words, no edges exist such that both of its endpoints lie in the same subset. From here on, for representing a bipartite graph, we would use the notation of $G = (U, V, E)$ where U and V are the 2 disjoint vertex sets,

[1]We use self loops in directed graphs while using the graphical representation of Markov chain, in Section 2.3.3

and E is the edge set. An edge $(u, v) \in E$ when $u \in U$ and $v \in V$. A bipartite graph, where each vertex in U has degree x, and each vertex in V has degree y is known as a (x, y)-*biregular* graph or (x, y)-regular bipartite graph.

2.1.2 Matching in Graphs

Definition. *Given a graph, $G = (V, E)$, a matching or independent edge set of the graph is an edge set $M \in E$ in which no edges share a common vertex. The matching of the largest cardinality possible is known as the maximum matching of the graph, and the size of maximum matching is denoted by $\nu(G)$.*

An important problem in graph theory is to find the maximum matching of a graph, that is to find the independent edge subset of a graph of maximum cardinality. This is known as the maximum cardinality matching problem, often referred to as the maximum matching problem.

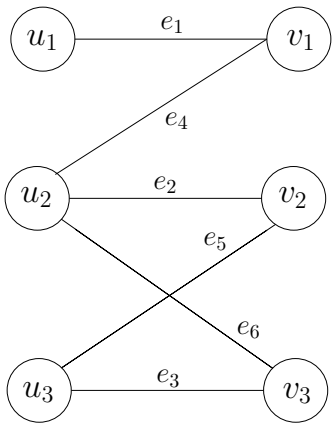


Figure 2.2: Example of a bipartite graph with 2 perfect and maximum matchings, $\{e_1, e_2, e_3\}$ and $\{e_4, e_5, e_6\}$. Note these 2 matchings are not disjoint.

A matching is said to be a *perfect* matching if all vertices have an edge incident on it belonging to the matching. *Disjoint* matchings are two or more matching sets in a graph which do not share an edge among themselves.

The work of this thesis revolves around calculating matching in bipartite graph in online setting, and we expand upon the constituent parts of the problem in the next few sections of this chapter.

2.2 Online Algorithms

Following on from the naive formulation of online problems in Chapter 1, we reintroduce online problems in this section. An online problem or a problem in *online setting* can be interpreted as follows: given a sequence of items as an input instance, an online algorithm outputs a decision after each input item, with the goal of optimizing an associated objective function. The algorithm cannot change its decisions after making them, and the decisions are final and irrevocable.

The key formulation of problems in online setting lies in the partial knowledge of the input instance and making final[2], irrevocable decisions on each input item without seeing the whole input sequence. Vis-à-vis the offline setting, where the algorithm is aware of the complete input instance in advance, and it can utilize the information of the whole input structure to optimize the associated cost function. We can formally define online algorithms and online problems as follows, except for a few classes of problems, such as exploration and navigation problems.

Definition (Online Algorithm)**.** *An algorithm which processes its input instance in a sequential piece by piece manner, and outputs immutable decisions for each input item is known as an online algorithm.*

We use ALG as a shorthand notation for the online algorithm from here on when the context suffices. There are many ways to characterize online problems formally, and we define it using a *request answer* framework [10], where items of input instance are presented as requests to ALG, and it subsequently provides an 'answer' to each of these requests.

Definition (Online Problem)**.** *An online problem can be formulated as a request answer exchange, comprising of request set R, answer set A, and objective functions,* $f_n = R^n \times A^n$.
Given an input instance $x = (x_1, x_2, \ldots, x_n)$ *of the problem, each item* $x_i \in R$ *is revealed incrementally to ALG, which presents an answer or decision* $d_i \in A$*, where* d_i *is dependent only on the input items revealed so far,* $x_{\leq i}$ *and the previous decisions,* $d_{<i}$*. The goal is to maximize or minimize the objective function* $f_n = (x_1, \ldots, x_n, d_1, \ldots, d_n)$ *after processing the whole input instance.*

[2]For some models of online problems, there is also an interest in making *semi* revocable decisions, which means, even if the algorithm can't go back and change the decisions, it can disregard or overwrite them in later stages.

2.2.1 Measuring Performance of Online Algorithms

The goal of the online algorithm is to make decisions such that the associated cost function gets maximized or minimized, depending on the problem. Since the discussion is centered around finding maximum matching in online settings, we define the rest of terms with respect to a maximization problem. Given input instance $x = (x_1, x_2, \ldots, x_n)$, we denote the value obtained by the objective function through ALG's decision as $ALG(x)$. While evaluating the performance of an online algorithm, the conventional metrics of time or memory(space) utilization are not used. A popular metrics of analysis of online algorithms is competitive analysis, introduced by Karlin et al. [49].

In competitive analysis, the performance of an online algorithm is judged by comparing the value of the objective function obtained by the online algorithm to that of presenting the same problem in an offline setting and getting an optimal offline solution. More precisely, we compare it with the value obtained by the best possible optimal offline algorithm, which we denote using OPT, and we denote the value of the objective function as $OPT(x)$. This gives us an idea about what ratio of output we lose when we present an offline problem in online setting. This ratio between the value obtained by ALG and OPT is known as the (asymptotic) *competitive ratio* of the ALG, and it is denoted using $\rho(ALG)$.

Definition (Asymptotic Competitive Ratio)**.** *We say that ALG is (asymptotically) ρ-competitive, or alternatively ALG achieves competitive ratio ρ, if for inputs x we have:*

$$ALG(x) \geq \rho \cdot OPT(x) - o(OPT(x)).$$

The (asymptotic) competitive ratio of ALG, denoted by $\rho(ALG)$, is the supremum over all ρ such that ALG is ρ-competitive, i.e.:

$$\rho(ALG) = \sup\{\rho : ALG \text{ is } \rho\text{-competitive}\}.$$

The online algorithm ALG is said to be $\rho(ALG)$-competitive if this ratio is bounded. From here on, we would denote $\rho(ALG)$ using ρ where the context is clear. Note that when the input generation or the decisions of ALG involve randomness, we take the ratio of expected values obtained by the

objective functions, rather than the standard function values computed directly from the objective functions.

Note that while analyzing an algorithm ALG for an online problem (in a maximization problem), we are primarily concerned with two results – the upper bound limits with respect to the problem, and the lower bound with respect to the algorithm being analyzed. The lower bound ρ_{LB} attained by the algorithm establishes that over any input sequence, the algorithm achieves at least ρ_{LB} times the value attained by the optimal algorithm OPT. In the same context, *the upper bound of an algorithm* is the best competitive ratio the ALG can attain, over all input instances. When these two, the lower bound of ALG, and the upper bound of ALG coincide, it illustrates that the analysis of ALG is *tight*.

The *upper bound of a problem* ρ_{UB} demonstrates that for any online algorithm, and any input sequence possible, the ALG can not perform better than ρ_{UB} times the value obtained by OPT. When the lower bound of an ALG matches the upper bound of the problem, we note that the performance of the algorithm is maximal for the problem and the bounds are tight.

By abuse of notation, we sometimes use ALG to refer to both the online algorithm and the value obtained the the objective function on the algorithm's decision when the context is clear. The same holds for OPT.

2.2.2 Modeling Graph Problems in Online Setting

Problems centered on graphs in online settings, particularly where the graph is an input can be formulated in many different ways. The underlying theme in the problems concerning *online graphs* is having knowledge of only the partial graph structure, where the graph gets built up as different sections of graphs present themselves in the input. In the same vein, the problem of finding maximum matching in bipartite graph in online setting, which has graph as its input, has different components of the graphs revealed incrementally in an online fashion. At any point, the ALG only knows about the unveiled part of the complete graph, with some vertices and edges still remaining.

Different scenarios lead to different ways of how the 'parts' or components of graphs are presented in a sequential manner. For example, inputs can be presented in the form of edges (denoted by a pair of vertices) from the edge set of the graph as input, which would build up the graph step-by-step. Another representation can be presenting the vertices of the graph along with information about the

neighborhood of the vertex.

Our thesis is concerned with matching in bipartite graphs, and we present a popular model of representing bipartite graph as input. Given a bipartite graph $G(U, V, E)$, the algorithm knows one side of bi-partition, U in advance. We label this as the offline vertex set. The other bi-partition, V is labeled as the online vertex set. Before the input sequence starts, only vertices from the offline set U are present. We follow the convention of U being the offline vertex set and V being the online vertex set for an online bipartite graph $G(U, V, E)$ throughout the thesis.

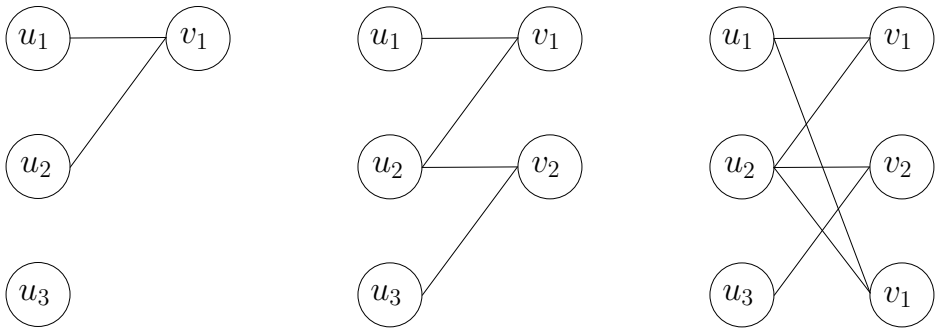


Figure 2.3: Example of an online graph building up through incoming online vertices in sequence $v_1,\ v_2,\ v_1$. Here, $U = \{u_1, u_2, u_3\}$ and $V = \{v_1, v_2\}$, and the neighbors of v_1 and v_2 are $N(v_1) = \{u_1, u_2\}$ and $N(v_2) = \{u_2, u_3\}$.

Each item of the input sequence (denoted by vector $\vec{\widehat{v}}$) consists of a pair, $(\widehat{v_i},\ N(\widehat{v_i}))$, where online vertex $\widehat{v_i}$ has 'a type' or is similar to a vertex in the online set V, in the sense that if $\widehat{v_i} \in \vec{\widehat{v}}$ and $v_j \in V$ are of the same type, they have the same neighbors. Mathematically, if $\widehat{v_i} \simeq v_j$ (are of the same type), then $N(\widehat{v_i}) = N(v_j)$. Along with the online vertex $\widehat{v_i}$, information about its neighbor in the opposite set U with which it shares an edge is also revealed. An example is presented in Fig 2.3. Online vertices occur in the sequence (v_1, v_2, v_1).

Sometimes, the vertices occurring online, in the input sequence are sampled from a distribution on the online vertices. In such cases, the graph formed in the online fashion, with offline vertex set U and these online occurring vertices are different from the base sample graph, which consisted of the U and the online vertex set V. We refer to the base graph, $G(U, V, E)$ as the *type graph* and the graph constructed online as the *realization graph*. We discuss these in more detail in sections where we introduce different input models for online bipartite matching.

Till now, we have presented an overview of what the problem of finding maximum matching

is, and how can this problem of bipartite matching be formulated in the online setting. Combining these two, we now present the definition of online bipartite matching.

Definition (Online Bipartite Matching)**.** *The online bipartite matching problem asks to find the maximum matching set in a bipartite graph $G(U, V, E)$, with U known in advance, and vertices (along with their incident edge set) having a type in V being revealed in an online manner. Upon the arrival of a vertex, the ALG decides to include (or not) one of the revealed edges in the matching set, such that the final matching is maximized.*

2.3 Input Models for Online Bipartite Matching

We have discussed how graph problems can be modeled in online setting – more specifically, *what* the input of online graphs can be. Now we discuss *how* these inputs are generated. Different ways of generating input instances for a problem are classified and studied under input models, which we discuss in this section. We also provide an overview of Markov chains, and introduce a new model of input generation using Markov chains.

2.3.1 Adversary

A popular representation of input generation for online problem is by conceptualizing a very powerful, all knowing adversary, which knows the online algorithm before hand, and gives each new input item such that the cost function becomes minimized relative to OPT (for a maximization problem). An *adversary* is a function or a strategy to generate input instances for the online problem (with respect to an online algorithm). One can imagine the adversary ADV and the algorithm ALG playing a game on the opposite sides. While the ALG makes decision to maximize the objective function, the ADV generates each input instance based on all previous input instances and ALG's decision on those input instances, such that the objective function is minimized relative to OPT. This is known as the *adversarial model* of input generation.

2.3.2 Randomness in Input Sequence

Another way of generating online input is by introducing randomness in the generated sequence, or sampling from a probability distribution.[3] Here, we primarily use *random variables* to characterize randomness in events. By convention, capital letters are used to denote random variables and expectation of the random variable is used to denote the weighted average of all values the random variable can take. The *expectation* of a random variable X is denoted by $\mathbb{E}(X)$, and if a X is sampled from a probability distribution $\mathcal{D}$, it is denoted using $X \simeq \mathcal{D}$.

Randomness in the input sequence can be introduced in different ways, for e.g. the ADV can choose a distribution on the input vertices, but the input sequence presented to the ALG can be a random permutation. This takes away the control from ADV from presenting the worst case sequence to ALG and is generally an easier model for ALG to achieve a competitive ratio, as compared to the adversarial model. Stochastic inputs can be generated from sampling the input from different well known distributions, or distributions that hold some property. These ways of generating inputs are commonly studied under stochastic input models. When inputs are generated in a random manner or stochastically, we compare the expected cost of the ALG to the expected cost of the OPT to analyze the performance of the online algorithm. Denoting the input sequence of random variables as $\vec{X} = (X_1, X_2, \ldots)$, the updated definition of (asymptotic) competitive ratio becomes:

$$\mathbb{E}_{X_1,X_2,\ldots}(ALG(\vec{X})) > \rho \cdot \mathbb{E}_{X_1,X_2,\ldots}(OPT(\vec{X})) - o(\mathbb{E}_{X_1,X_2,\ldots}(OPT(\vec{X})))$$

A famous stochastic process Markov chain, which we use to sample online vertices introduced in a new model is discussed next.

2.3.3 Markov Chains

Markov chains (we consider only discrete time Markov chains in this thesis) are stochastic models used to generate a sequence of random variables. The sequence of random variables holds the *Markov* property, which limits the dependence of the probability of the current event only on

[3] While stochastic and random represent the same thing, we present popular, well known distributions under stochastic setting and use random models for general, arbitrary permutations and orderings.

the probability of immediately preceding previous events. Another way of representing the Markov property is by emphasizing the 'memorylessness' of the stochastic process – given the current state, the future states of the Markov chain are independent of its past states.

Given a Markov chain $\mathcal{M}$, the *state space* $S = \{x_1, x_2, \cdots x_n\}$ represents the set of values which the random variable sampled from $\mathcal{M}$ can attain. Obtaining k samples, we get a k-length sequence of random variables, $X_1, X_2, \cdots X_k$, which satisfy the Markov property:

$$P(X_k = x_k \mid X_1 = x_1, X_2 = x_2, \ldots X_{k-1} = x_{k-1}) = P(X_k = x_k \mid X_{k-1} = x_{k-1}).$$

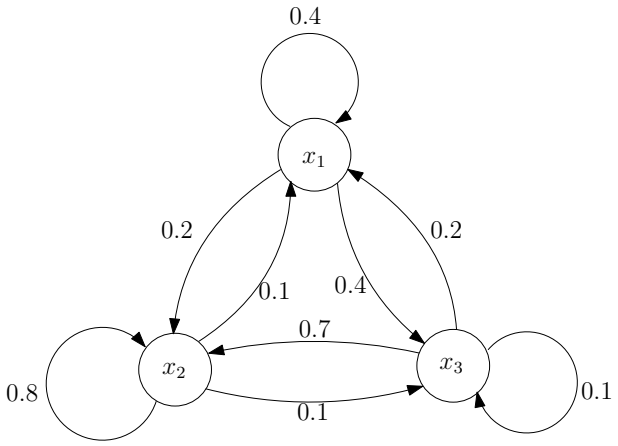


Figure 2.4: Example of a Markov chain on 3 states, x_1, x_2 and x_3.

$$\begin{bmatrix} 0.4 & 0.2 & 0.4 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.7 & 0.1 \end{bmatrix}$$

Figure 2.5: Stochastic Matrix of the corresponding Markov chain, $M[i][j] = P(X_{t+1} = x_j \mid X_t = x_i)$.

The distribution of the first random variable among the sampled sequence is given by an initial distribution, denoted using π. Baring the first random variable, the value obtained by any subsequent sampled random variable is conditionally dependent on the value of its preceding random variable only. This conditional probability is given by the transition *stochastic matrix*, M of the Markov chain. The stochastic matrix is a square matrix of $n \times n$ dimension, which gives the probability of transition from one state to other. The transition probability between states x_i and x_j is given by $M[i][j]$. We refer to the following Markov chain by $\mathcal{M}(\pi, M)$.

Definition (Markov Chain)**.** *A discrete time Markov chain $\mathcal{M}(\pi, M)$ on state space $S = \{x_1, x_2, \cdots x_n\}$ is a stochastic process used to generate a sequence of random variables, $X_1, X_2, \ldots, X_t$ that satisfies the Markov property, which refers to the fact that the future state depends only on the present state,*

not on the sequence of past states. Formally:

$$P(X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0) = P(X_{t+1} = x_{t+1} \mid X_t = x_t),$$

for all $t > 0$, where t represents the discrete time steps at which a random variable is sampled, π is the initial distribution for sampling X_1, and M represents the transition matrix.

We assume the transition probabilities of Markov chains which we deal with don't change with time. A Markov chain is time-homogeneous if the transition between two states is independent of k, $P(X_k = x \mid X_{k-1} = y) = P(X_{k+1} = x \mid X_k = y)$.

A famous and well studied stochastic process, random walk can be simulated using Markov chains. *Random walks* on a mathematical space is a sequence of random variables that reflect random subsequent steps taken on the space starting from an initial position. Random walks on graphs can be thought of as starting from an initial vertex, and then traveling to a next vertex (or staying at the same vertex) in discrete time steps. The transition from current vertex to the next is described by the probability of jumping between these 2 vertices.

A Markov chain essentially replicates the same movement, if the graph is strongly connected (there might be a probability of moving between unconnected components of the graphs, but we restrict movement only between described state space in Markov chains). The vertex set of the graph represents the state space of the Markov chain, and the probability of 'walking' from one vertex to another can be emulated by its stochastic matrix. A popular class of random walks is *lazy random walk*, in which the walk stays at the same vertex with probability $\frac{1}{2}$, and moves to its neighbors uniformly with the combined remaining probability of $\frac{1}{2}$.

We generalize this class of random walks, by defining a parameter p, which denotes the probability of remaining in the same state. The neighboring vertices can be uniformly reached with the remaining probability of $1 - p$. In the case of Markov chains with n states, this can be defined using a stochastic matrix, where $M[i][i] = p$ and $M[i][j] = \frac{1-p}{n-1}$, for all $i, j \in [n]$. We exhibit that the Markov process can start at any n states uniformly, and the initial probability distribution is $\pi = \mathcal{U}[n]$, where $\mathcal{U}[n]$ denotes the uniform discrete probability distribution among n states, the probability of each state being $\frac{1}{n}$.

With this, we have sufficient information to define the Markov chain based input model, which we present in the next section.

2.3.4 Summary of Input Models for Online Bipartite Matching

We summarize a few different models in which inputs can be generated for the focal problem here – 'online bipartite matching'. Note, for $G(U, V, E)$, U is the offline vertex set and known in advance to the algorithm, and V is the online vertex set, and online vertices with type in V are the input requests.

(1) Adversarial model: The input is generated using an adversary ADV, under the circumstances that the algorithm ALG only knows U before the input begins, and has no knowledge of what the 'types' of online input vertices in V and their neighborhood E would be. In other words, ALG is not familiar with the input graph structure beforehand. The ADV chooses the worst possible graph G and generates the worst possible input sequence.

(2) Random order model: Similar to the Adversarial model, the ADV chooses the worst graph G for the problem, but it can't choose the order of input vertices, i.e. the ordering of the online sequence is randomly generated, and can be any permutation of the possible input sequences.

(3) Stochastic model: Similar to the Random model, ADV chooses and fixes the type graph G for the problem before the input requests begin. Each input vertex $\widehat{v_i}$ is chosen from a distribution $\mathcal{D}_i \in \vec{\mathcal{D}}$, where $\vec{\mathcal{D}}$ is a vector of probability distributions over the online vertex set V. Whether $\vec{\mathcal{D}}$ is known to ALG in advance or not divides the problems into different settings of known distribution and unknown distribution models. We focus on models in which ALG has a priori knowledge of the distributions because we want to use this beforehand statistical knowledge to maximize the size of the matching set.

A well known model studied under this scenario is the Independently and Identically Distributed (IID) model. We discuss here the Known IID model, but its 'unknown' counterpart, the Unknown IID model is also well studied and researched. Another model of sampling from a distribution can be sampling from a Markov chain known beforehand to the ALG, which we introduce in this thesis. We discuss these two models below.

(a) Known IID model has all vertices sampled from a fixed distribution $\mathcal{D}$ in an independently and identically distributed fashion, which means each vertex is sampled from the same distribution (identical part), and choosing each online vertex is independent of any other decision (independent part).

(b) Known Markov chain model, introduced for the first time in this thesis presents a way of sampling vertices from a Markov chain $\mathcal{M}$ for the online bipartite matching problem.

Definition (Known Markov Chain Model)**.** *Online bipartite matching in graph $G(U, V, E)$, under the Known Markov chain model asks to find the maximum matching set, where online vertices having a type in V are sampled from a Markov chain $\mathcal{M}$ distributed over V.*

We want to highlight two points related to the different input modeling of the problem. First, under stochastic settings with known distributions, the algorithm has advance knowledge about U similar to any other input model stated above. As it has information about the distribution from which online vertices are sampled from, it also knows all possible vertices in V (and its edge set between vertices from V to U), over which the samples are distributed. Hence, the ALG has a priori knowledge about both vertex sets U and V, and the edge set E. This can be reorganized to state that in an online bipartite matching problem, under known distribution (stochastic) settings, the ALG has beforehand knowledge about the *type graph* of the problem, $G(U, V, E)$. The final graph formed by U and vertices obtained by sampling from the distribution over V is known as the *realization graph* of the problem. Note that in Adversarial and Random order model, only one graph is present in the problem, the 'input graph' as the graph chosen by ADV is presented to the ALG incrementally.

Second point, it can be observed that problem of maximizing matching under given stochastic information about the graph makes it easier. More information about the structure of graphs makes the process of decision making of whether to include an edge in matching or not more optimized. Another point to be highlighted, it can be seen that the Random order model is weaker than the Adversarial model because in the latter, the adversary has power of choosing the worst case input instance, which it loses in the former. Because ALG obtains at least ρ_{Adv} in Adversarial model, for all input instances, it does so even in expectation over a random input order. Combining these, we

can informally state that from an algorithm designing perspective, Known IID model is easier than Random order model, and Random order model is easier than Adversarial model. Informally, we can state the following theorem:

Theorem 1 (Competitive Ratio under Input Models)**.**

$$\rho_{Adv} \leq \rho_{RO} \leq \rho_{Known-IID}$$

For now, because of the lack of extensive research on Markov chain model, and Markov chains being a very general model and a part of a very large family of generating stochastic processes, it is difficult to analyze where this model places in the above model concretely.

We finish off the Preliminaries chapter by summarizing the above discussion in stating the problem at the crux of this thesis – 'online bipartite matching under Markov chain model'.

2.4 Online Bipartite Matching under Markov Chain Model

Combining all our previous segments, we can now formally state the problem of *online bipartite matching under Markov chain model*. Online bipartite matching under Markov chain model refers to the problem of solving the maximum cardinality matching problem in bipartite graph in an online setting, where the input – online vertices are generated using a Markov chain on the online vertex set. The online algorithm, ALG makes online decisions on whether to include an edge between the online vertex and its available neighbor in the final matching, such that the size is maximized.

We describe our problem as follows:

Given a bipartite graph $G = (U, V, E)$, with U and V being the offline and online vertex set respectively, which we refer to as the *type graph* of the problem. A Markov chain $\mathcal{M}(\pi, M)$, on the state space V generates a k length sequence of online vertices, denoted by $\vec{\widehat{v}} = (\widehat{v}_1, \widehat{v}_2, \cdots \widehat{v}_k)$. Each online vertex, $\widehat{v}_i$ has a type in V, denoting that $N(\widehat{v}_i) = N(v_j)$, for some $v_j \in V$. At the arrival of $\widehat{v}_i$, along with the information of its incident edges to vertices in U, the ALG decides to include (or not) one of these edges to the matching set maintaining that the matching set is independent throughout. ALG can include at most one edge for each online vertex, otherwise there would be

2 or more edges with the current online vertex being common. The objective of the problem is to maximize the size of matching set. The graph formed, between U and (after encountering) all online vertices is referred to as the *realization graph* of the problem. The efficiency of ALG is computed by comparing the size of matching formed by ALG's decisions to the size of matching computed in the realization graph under offline settings by an optimal offline algorithm OPT.

Currently, the problem for general type graphs and general Markov chains is quite vast and has no specific results. For this thesis, we discuss the problem of online bipartite matching for specific classes of type graph and Markov chains as input. Particularly, we study matching in bipartite graphs which are biregular, specifically $(2, 2)$-biregular graphs. For input generation, we employ Markov chains behaving similarly to the phenomena of lazy random walk as discussed in Section 2.3.3.

Chapter 3

Literature Review

Graphs are a foundational concept in mathematics and computer science, serving as a versatile framework for modeling relationships and interactions in a wide range of domains. From social networks, where users and their connections are represented as nodes and edges, to communication systems that model routers and data links, graphs provide an elegant framework for understanding complex structures. Graphs can be modeled to capture the complexities of protein-protein interactions and gene regulatory networks in biology, providing insights into molecular mechanisms and cellular functions. In physics and chemistry, graphs represent lattice structures and molecular bonds, enabling the study of material properties and chemical reactions. In machine learning, graph-based frameworks power advancements such as recommendation systems and graph neural networks, allowing for the processing of data with intricate, non Euclidean relationships. Meanwhile, in theoretical computer science, graphs drive advancements in areas like algorithm design for shortest paths and network flows, complexity theory through reductions and completeness problems, and automata theory via state transition models.

This thesis narrows its focus to matching problems in bipartite graphs, a fundamental area within graph theory with practical and theoretical significance. Matchings in bipartite graphs have wide-ranging offline applications, such as assigning tasks to workers, pairing students to projects, or matching organs to patients in transplant programs. A few prominent problems include stable marriage (matching) problem [32], stable roommates problem [44], optimal kidney exchange [5, 4], resident doctor matching [66], etc.

In such cases, the entire problem instance is known upfront, allowing algorithms like the Hungarian or Hopcroft-Karp to compute optimal or near optimal solutions efficiently. In contrast, online bipartite matching focuses on scenarios where input arrives sequentially, and decisions must be made immediately without knowledge of future data. This is critical in applications like online advertising, where ad slots and advertisers are matched in real time, or ride sharing platforms, where passengers and drivers are dynamically paired. Online matching algorithms strive for competitive performance, measured against the ideal offline solution, making their applications indispensable in dynamic, real-time systems.

In this chapter, we present a brief history of the problem of matching in graphs and some important results. We first present the problem in offline setting, showing important theorems, algorithms and results in this problem in the past century. We then survey some recent developments in the topic of online matching, and summarize important benchmarks in different input models discussed in Chapter 2.

3.1 A Short History of the Matching Problem

Matching has been a central problem in graph theory, with its roots lasting back to early 20th century. In bipartite graph, matching has found applications in a spectrum of fields, particularly in economics [1, 18, 67], wherein the theory of matching markets resources are allocated to respective assignees in an optimal fashion.

Early results on matching in graphs were presented by Kőnig [60], who stated that the cardinality of maximum matching in a bipartite graph is equal to the size of minimum vertex cover. A *vertex cover* of a graph is the subset of the vertex set such that it includes at least one endpoint of every edge of the graph. The result, known as *Kőnig's theorem* was also independently discovered by Egerváry [23] for the generalized case of graphs with weighted edges.

In general, the problem of finding the smallest vertex cover of an arbitrary graph is $\mathcal{NP}$-hard, and also hard to approximate. The vertex cover set of a graph is at least as big as any matching set of the graph, because the vertex cover must contain at least one vertex from each matching edge. But in the case of bipartite graphs, the size of minimum vertex cover is exactly equal to the size of

maximum matching.

3.2 Offline Bipartite Matching

Offline bipartite matching can be studied under several settings, based on the nature of the underlying graph structure. We first present results on matching in unweighted bipartite graphs, and then generalize the results to vertex and edge weighted bipartite graphs. Since the problem of matching in general graphs also lie in the same domain, we briefly just state the results in the closing remarks of the section, without going into too much depth.

3.2.1 Unweighted Bipartite Matching

Matching in both, bipartite and general arbitrary graphs can be computed optimally in $\mathcal{P}$ complexity class. In Berge [7], the author introduced the concept of augmenting paths which serves as the framework for many algorithms in calculating the maximum matching in bipartite graphs. For a graph $G(V, E)$ and matching M, an *alternating path* is a simple path of edges present alternatively in M and $E \setminus M$. An *augmenting path* is an alternating path with exposed end-points, where the starting and ending edges of the path are not in M.

Berge's theorem states that a matching M is maximum iff it does not admit an augmenting path with respect to M. If it does admit an augmenting path, then flipping edges in the augmenting path gives a matching, i.e. the edges in the augmenting path but not in M are also a matching, and of size 1 greater than size of matching M. Thus, matching in graphs can be calculated by initially starting from an empty matching set and finding augmenting paths to update the matching set, until the matching doesn't admit an augmenting path.

In bipartite graphs, augmenting paths can be efficiently detected by using Breadth First Search (BFS) algorithm. Starting from a vertex s, which has no edges from matching set incident on it, we search among the neighbors of s and either find another vertex t which has no edges in the matching set incident on it, giving an augmenting path. Or if the vertex t has an edge in the matching incident on it, let's say with vertex u, then we find an augmenting path from u, along the path $s - t - u$, in which the edge between s and t is not in the matching, and the edge between t and u is present in

the matching. To check if an augmenting path can be found with respect to a matching, we run this BFS step for all vertices $s \in V$, which takes $O(|V|)$ time and the BFS algorithm takes $O(|E|)$ time, making the time complexity of the process to be $O(|V||E|)$.

The problem of finding maximum matching in bipartite graphs can be reduced to another closely related problem, of finding maximum flow in a flow network. A *flow network* is a directed graph, where each edge has a capacity, or the maximum flow possible through the edge. The graph has two special vertices, *source* and *sink*, which only has outgoing and incoming flow respectively. The objective of the problem is to find the maximum amount of flow that can pass through the network, respecting the edge capacity constraints. A bipartite graph $G(U, V, E)$ can be converted to a flow network by adding auxiliary source – s and sink – t vertices on the opposite sides of the bi-partition, such that there are outgoing edges from s to all vertices in U, and similarly, incoming edges from all vertices in V to t. The capacity of all edges in the newly created flow network is 1.

In 1956, Ford and Fulkerson [28] introduced the Ford Fulkerson method, a greedy procedure that calculates the maximum flow in a flow network by finding an augmenting path from source to sink. In general graphs with integral edge capacity, finding augmenting paths takes $O(|E|)$ time, and the procedure takes $O(f \cdot |E|)$ time overall, where f is the maximum flow possible. This is because, each time an augmenting path is found, flow increases by at least 1. Edmonds-Karp Algorithm, introduced in Edmonds [21], an implementation of the Ford Fulkerson method computes the maximum flow in a network in $O(|V||E|^2)$ time. Dinic's Algorithm [14], based on level graphs and blocking flow, runs in $O(|V|^2|E|)$.

Both the Edmonds-Karp Algorithm and Dinic's Algorithm are used to calculate the maximum flow in flow networks and can be used to calculate the maximum matching for a bipartite graph, modified to be a flow network. Another algorithm for calculating maximum flow in bipartite graph, Hopcroft-Karp Algorithm, introduced independently in [38] and [52] has a run time of $O(\sqrt{|V|}|E|)$.

While the focus is on matching in bipartite graphs, a closely related and quite important theorem discussing matching in the case of general graphs cannot be overlooked. In the case of non bipartite graphs, augmenting paths can be missed due to the presence of odd length cycles in the graphs ([53] illustrates an example for this scenario). In Edmonds [20], Edmonds gave the

famous Blossom Algorithm for calculating the maximum matching in arbitrary graphs. It works by finding *blossoms* – odd length cycles in the graphs, and contracting it to a single vertex and continuing the search of finding an augmenting path in the graph. This is a polynomial time algorithm, with its time complexity being $O(|V|^2|E|)$, and this paper introduced the concept of $\mathcal{P}$ time complexity/algorithms.

A faster algorithm for calculating maximum matching in arbitrary graphs runs in $O(\sqrt{|V|}|E|)$ time, given by Micali et al. [65] (rectified in [71]). Similar bounds can be achieved by algorithms in Blum et al [9], Gabow et al [31], etc.

Weighted Version

The weighted versions of bipartite matching are generalizations of the unweighted bipartite matching problem, where either vertices or edges have weights associated with them, and the objective is to find the matching set that maximizes the summation of these quantities. The model of matching problem discussed above can be more precisely defined as maximum cardinality matching, where the goal is to find the matching set of largest cardinality possible. We introduce the two different versions next.

3.2.2 Vertex Weighted Bipartite Matching

In the vertex weighted bipartite matching problem, the target is to find a matching set, such that the sum of weights of vertices of the edge in the matching set is maximized. Dobrian et al. [15] and Halappanavar [37] gave a $O(|V||E|)$ algorithm for computing maximum vertex weighted matching in bipartite graphs, which works by dividing the graph into two one side weighted bipartite graph and solving the problem of maximum weighted matching on it. The two matchings are combined using the *Dulmage–Mendelsohn* decomposition [17].

Spencer et al. [68] gave a $O(|V|\log(|V|)|E|)$ for solving the problem in general graphs. The theses of Halappanavar [37] and Al-Herz [3] cover exact and approximation algorithms for this problem in detail.

3.2.3 Edge Weighted Bipartite Matching

Another well studied model of this problem is maximum weight matching problem. Given a graph with edge weights, the goal is to find a matching set such that the sum of edge weights is maximized. The maximum cardinality matching problem is a special case of the maximum weight matching problem, with all the edge weights being 1. The Blossom algorithm [20], discussed above can find the maximum weight matching for both bipartite and non-bipartite graphs in $O(|V|^2|E|)$ time.

The maximum weighted matching problem in bipartite graphs is popularly known as the *assignment problem*. The Hungarian method, introduced by Kuhn in [58] gave an $O(|V|^4)$ procedure of finding the solution to this model, which was modified upon to achieve $O(|V|^3)$ time complexity independently by Edmonds and Karp [22] and Tomizawa [70].

Johnson [47] gave an $O(|V| \log_d |V||E|)$, where $d = 2 + \frac{|E|}{|V|}$ algorithm for weighted bipartite matching problem, which was improved to $O(|V|^2 \log |V| + |V||E|)$ by Fredman and Tarjan [29] by the use of Fibonacci heaps. A similar bound is achieved for maximum weighted matching in general graphs given in Gabow [30].

Duan and Pettie [16] surveys and presents important results on linear time approximation algorithms on finding maximum cardinality matching and maximum edge weighted matching problems.

3.3 Online Bipartite Matching

The focus of the work in this thesis is centered around calculating the size of matching in online bipartite graphs. In particular, we focus on stochastic setting, where the occurrence of online vertices have a Markovian dependency among them. As discussed in Section 2.3.4 of the previous chapter, different input models under which the online bipartite matching problem is studied are – Adversarial model, Random order model, Known IID model (under stochastic models). We once again discuss this problem under different input models for unweighted and weighted bipartite graphs, and provide some results pertaining to the lower bounds and upper bounds of the problem setting.

Some literature surveys covering important results in online matching are as follows: An excellent literature review by Mehta [63] covers important results and techniques in online matching and ad

allocation. For a more recent survey, which introduces more different modeling of the online matching problem, we direct the readers to refer to Huang et al. [42]. Other works reviewing online matching are [27] and [34]. Devanur and Mehta [13] is a recent work, covering online matching from an ad allocation, auctions and matching market perspective.

3.3.1 Unweighted Online Bipartite Matching

As implied in the model name, the underlying bipartite graph is unweighted, or it can be assumed that each edge carries the same weight. Under this graph, the objective is to maximize the size of the matching set.

Adversarial model

In the unweighted (online) bipartite graph, $G(U, V, E)$, the online vertices with type in V are generated by an adversary, with the algorithm making decisions on matching the current online vertex with some vertex in U (or not), with the goal being to maximize the size of matching in the generated realization graph. The size of matching is compared with the maximum possible matching in the final complete graph, given all online vertices are already received.

In the realm of deterministic algorithms, a natural greedy algorithm, Deterministic Greedy ($Greedy_{Det}$), which matches the current online vertex to an available offline vertex from a predetermined order of offline vertices achieves a competitive ratio of $\rho(Greedy_{Det}) = \frac{1}{2}$. $Greedy_{Det}$ creates a maximal matching, and any maximal matching is at least $\frac{1}{2}$ times the optimal matching. No deterministic algorithm can do better than $\frac{1}{2}$, and hence this bound is tight in the deterministic settings, as depicted in Fig 3.1.

A randomized greedy algorithm, Random, which matches the online vertex to a random available neighbor also achieves similar competitive ratio of $\frac{1}{2} + o(1)$. This ratio is tight and can be achieved by repeating the instance of graph used in the case of $Greedy_{Det}$ [51].

In their seminal work, Karp, Vazirani, Vazirani [51] introduced another randomized algorithm, which achieves a competitive ratio of $1 - \frac{1}{e}$. The algorithm, known as Ranking Algorithm works by choosing an arbitrary preference ordering of offline vertices U randomly from all possible permutations, and fixes it. On arrival of an online vertex, it is matched with the highest ranked

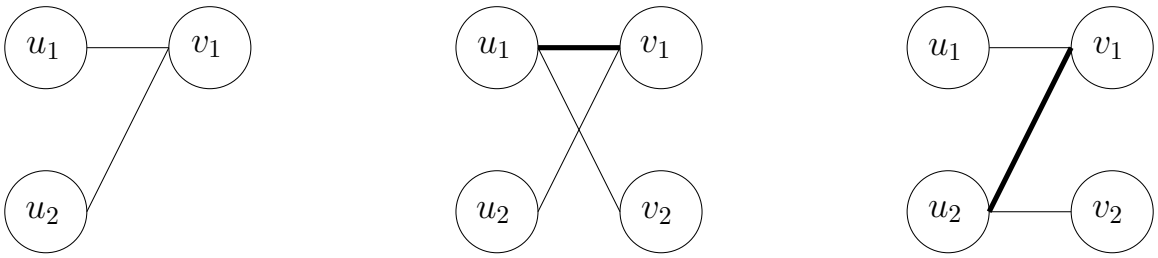


Figure 3.1: No deterministic algorithm can perform better than $\frac{1}{2}$. The ADV generates $v_1, N(v_1) = \{(u_1, v_1), (u_2, v_1)\}$ as the first input. If ALG matches v_1 with u_1, the next input is $(v_1, N(v_2) = \{u_1, v_2\})$ as shown in the second graph, otherwise if ALG matches v_1 with u_2, the next input is $(v_2, N(v_2) = \{u_2, v_2\})$, restricting the maximum possible matching to 1 in online setting. $OPT = 2$ for each case.

available offline vertex according to the ranking order.

The original paper had a mistake in the proof, independently observed by Goel and Mehta [35] and Krohn and Varadarajan [57]. [35] also provided a rectified version in their paper, and other proofs came along in Birnbaum et al. [8], Devnaur et al. [13], Eden et al. [19], etc. The original paper also proved that the bounds achieved by RANKING is tight, i.e. no algorithm can perform better than $1 - \frac{1}{e}$ in online adversarial setting.

Random order model

Karp et al. [51], and subsequently, Goel et al. [35] studied the problem of online bipartite matching, where the type graph is generated by the adversary but the online vertex request is generated in random order. They showed that a deterministic and greedy algorithm DETERMINISTIC GREEDY, which matches the current request with an available offline vertex arbitrarily achieves $1 - \frac{1}{e}$ ratio, by showing that the DETERMINISTIC GREEDY with random inputs and RANKING with adversarial inputs are dual of each other [27].

Karande et al. [48] showed that RANKING achieves a competitive ratio of 0.653, strictly better than $1 - \frac{1}{e}$. This bound was improved to 0.696 by Mahdian et al. [61] which uses a computer based technique to solve a family of strongly factor revealing LPs. An experimental upper bound on the modified RANKING algorithm of 0.724 was obtained by Chan et al. [12].

The upper bound in this model for deterministic algorithms is 0.75 and for random algorithms is 0.833 [35].

Known IID model

Under stochastic models, most research has been done under the Known IID setting, in which the online vertices are generated from an independent and identical distribution. This model was introduced by Feldman et al. [26]. In this model, the algorithm knows the offline vertex set U, and also a distribution $\mathcal{D}$ from which the online vertices V are sampled. Combining these two, the algorithm has knowledge of the underlying *type graph* and distribution $\mathcal{D}$. The paper discussed two algorithms for this model.

Suggested Matching, a relatively simpler algorithm, finds an optimal matching in the base (type) graph, and matches the online vertex request with respect to it, i.e. if the corresponding offline vertex with respect to the optimal matching for the current request is available, then it gets matched. This algorithm achieves a $1 - \frac{1}{e}$ competitive ratio, and the analysis for the algorithm is tight.

Another algorithm, Two Suggested Matching (Non Adaptive) finds 2 large disjoint matchings, M_1 and M_2 in the type graph, and uses these 2 matchings to guide the decision making process on the incoming online vertices. On the first arrival of a vertex, say v^*, it is matched in accordance to its neighbor in $M_1(v^*)$ if available, and on its second occurrence, it is matched in accordance to its neighbor in $M_2(v^*)$. This algorithm achieves competitive ratio $\frac{1-\frac{2}{e^2}}{\frac{4}{3}-\frac{2}{3e}} \approxeq 0.67$ with high probability, and this analysis is tight. They also proved an upper bound of 0.98 for this model.

Bahmani et al. [6] improved this bound to 0.699 by adding a step of preprocessing the graph and computing the 2 matchings. They also established an upper bound of 0.902 for the model.

In Manshadi et al. [62], the authors presented a more efficient method to find 2 disjoint matchings in the type graph (using fractional matching), and the same steps to match the online vertices. This improves the bound to 0.684. They also gave an adaptive version of the algorithm, Two Suggested Matching (Adaptive), which changes how the online vertices are matched. For each occurrence of v^*, it checks both of its neighbor in accordance with matching M_1 and M_2 to match in that order (given they are available to match). The non adaptive version doesn't check both matchings for each occurrence of v^* and creates a matching of smaller size as compared to the adaptive version. The competitive ratio achieved by this algorithm is 0.702 and no algorithm under this setting can do better than 0.823, improving upon the previous upper bound. As the Known IID model is a special

case of Random order model, this also improved upon the upper bound of $\frac{5}{6}$ of the latter model.

Jaliet and Xu [45] and Huang and Shu [40] proposed algorithms which improved the competitive ratio to 0.706 and 0.711 respectively. Huang et al. [41] further improved the bound to 0.716, the best known result in this model till now.

Summary of Algorithms

Models	Deterministic Lower	Random Lower	Deterministic Upper	Random Upper
Adversarial model	0.5 [p. 26]	$1 - \frac{1}{e}$ [51]	0.5 [p. 26]	$1 - \frac{1}{e}$ [51]
Random order model	$1 - \frac{1}{e}$ [35]	0.696 [61]	0.75 [35]	0.823 [62]
Known IID model	0.699 [6]	0.716 [41]	0.823 [62]	0.823 [62]

Table 3.1: Table summarizing lower and upper bounds in online bipartite matching under different input models.

Weighted Version

Two generalizations which follows from the unweighted bipartite models are – vertex weighted and edge weighted online bipartite matching.

3.3.2 Vertex Weighted Online Bipartite Matching

In online vertex weighted bipartite matching, the objective is to maximize the sum of matched offline vertices. Note, vertices in U only have non negative weights. The unweighted bipartite model is a special case of this, under which all offline vertices have an equal weight of 1.

Adversarial model

This model was introduced in Aggarwal et al. [2], and a generalized, weighted version of RANKING algorithm, PERTURBED GREEDY was given which achieves $1 - \frac{1}{e}$ competitive ratio for general vertex weights in Adversarial model. This bound is tight for this model, where the upper bound follows from the unweighted case.

Random order model

In the Random order model, introduced in Huang et al. [43], the authors showed that a generalized version of the RANKING algorithm achieves a ratio of 0.6534 using a randomized primal-dual framework, beating the $1 - \frac{1}{e}$ bound in the adversarial setting. Jin et al. [46] improved the algorithm analysis to obtain a competitive ratio of 0.6629, and showed an upper bound of 0.6688 on the algorithm.

Known IID model

Previous results for vertex weighted matching in Known IID model were under additional constraints of integral arrivals, i.e. the expected number of online vertices of each type present in the input sequence is integral. Under this constraint, a competitive ratio of 0.667 [36] and 0.725 [45] have been achieved.

Jin et al. [46] and Huang et al. [41] gave algorithms for this model with competitive ratios of 0.662 and 0.716 respectively. As of now, the latter is the best known result in this field.

3.3.3 Edge Weighted Online Bipartite Matching

As discussed in the offline setting, edge weighted online bipartite matching problem asks to find a matching set where the sum of edge weights are maximized.

Adversarial model

In the Adversarial model, the online edge weighted bipartite matching is not competitive without constraints as shown in Fahrbach and Zadimaghaddam [25], but variations under this model are studied. Given the edge weights range from $[L, U]$, a deterministic algorithm can achieve a tight competitive ratio of $\phi + 1$, where $\phi = \frac{U}{L}$ [10].

More commonly, the edge weighted model is studied under the *free disposal assumption*, which has applications in the advertisement model discussed in later Section 3.4. Under free disposal settings, introduced in Feldman et al. [26], the algorithm can revoke previously accepted edges between the online and offline vertices. Fahrbach et al. [24], using a new technique, online correlated

selection presents a 0.5086-competitive algorithm for this model under the free disposal assumption.

Random order model

In Random order model, Kesselheim et al. [54] introduced a $\frac{1}{e}$ competitive optimal algorithm for edge weighted bipartite matching. This improved the $\frac{1}{8}$ bound algorithm given by Korula et al. [55], and matches the upper bound for the problem, as stated in Buchbinder et al. [11].

Known IID model

Yan [73] presents a 0.645 competitive algorithm for edge weighted online bipartite matching under the stochastic settings and Huang et al. [41] presents an 0.703 upper bound for this model.

3.4 Applications of Online Bipartite Matching

Bipartite matching problems in online settings are ubiquitous and important in many disciplines. They find applications in online advertisements, crowd sourcing applications, food delivery services, matching children with schools, doctor residency matching programs, online dating services, etc.

As discussed in Chapter 1, in online advertising, particularly in the domain of ad allocation, advertisers and available ad slots (impression) form two sets in a bipartite graph, and each ad request needs to be matched with the best ad slot in real-time. As each request arrives, the advertisement platform must quickly decide the best match based on available information, with the goal of maximizing revenue or relevance, often expressed computationally in the size of the matching set's weight.

Another application is in ride sharing services, where drivers and passengers are dynamically matched as requests come in. This forms a bipartite graph where the drivers are matched to passengers based on proximity and availability, optimizing for both fairness and efficiency. Note that a better conceptualization of this problem would be to take into fact that both sides of the bi-partition can arrive online, it is not necessary that the cab drivers (or the passengers) are already present when the matching starts. Another constraint can be introduced in terms of reusable resource models – in which an offline vertex can be used for matching again after a time period (or time steps in discrete

modeling). This can be used to model that a cab driver can once again start servicing customers (start matching with them) once they have completed a ride. Such problems and models are described in [59, 39, 72, 33].

In job matching platforms (such as freelancer websites), crowd sourcing platforms and service based marketplaces like TaskRabbit and Upwork, employers post tasks while freelancers offer services. As new job offers arrive, an online matching algorithm pairs the most appropriate freelancers to tasks in real-time. These applications are typically modeled as online bipartite matching problems, where the challenge is to make decisions with limited information and optimize performance against an ideal offline solution, often using competitive analysis for benchmarking.

We discuss two applications specific to the field of online advertisement, which follow naturally from the weighted bipartite matching model.

With applications in ad allocation, two popular formulation of online bipartite matching are – DisplayAds and Adwords problem, which we discuss next.

A popular characterization since the introduction of free disposal assumption in the edge weighted online bipartite matching, the *Display Ads* problem was introduced in [26], where the objective is to maximize the total weight of the matching under the caveat that each offline vertex u has an integral capacity c_u which denotes the maximum number of edges of online vertices it can admit or be matched to. This generalizes the notion that an advertiser can display ads to up to c_u number of impressions.

Another modification of the problem introduced in [64] is the *Adwords* problem, where each offline vertex u has a budget B_u and each edge has a bid cost $bid_{u,v}$. Each matching depletes the budget of u by the corresponding bid value of the matched edge, and an offline vertex can continue matching till its budget is not depleted. The other models, unweighted and vertex weighted bipartite models are special cases of this problem. In case of unweighted matching, the budget of all offline vertices is 1, and each bid or the edge weighs. 1. The vertex weighted model again has the weight of each offline vertex equal to its budget B_u, and the bids for an edge $bids_{u,v} = B_u$, for all online vertices v.

3.5 Online Algorithms with Markovian Input

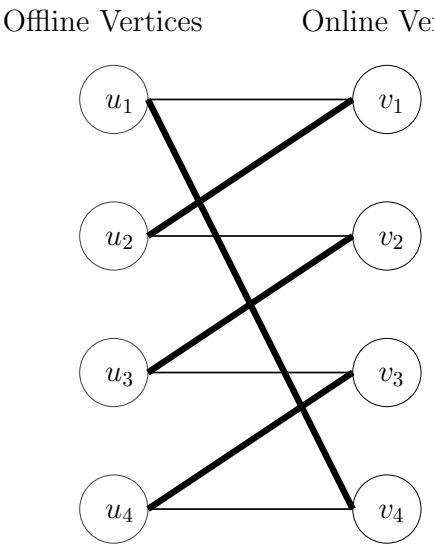
So far, we have introduced and summarized research related to different input models for online bipartite matching. Now, we initiate studying the new model introduced in this thesis – 'Online Bipartite Matching in Markov Chain Model'. Previous work concerning online algorithms under Markovian dependency has been studied by Karlin et al. [50], in which the authors studied the problem of paging under the assumption that page requests are generated using a Markov chain.

We present our work on online matching, precisely on designing, calculating, and analyzing algorithms for online matching in $(2, 2)$-regular bipartite graphs under the family of parameterized 'lazy random walk' based Markov chains in the next two chapters.

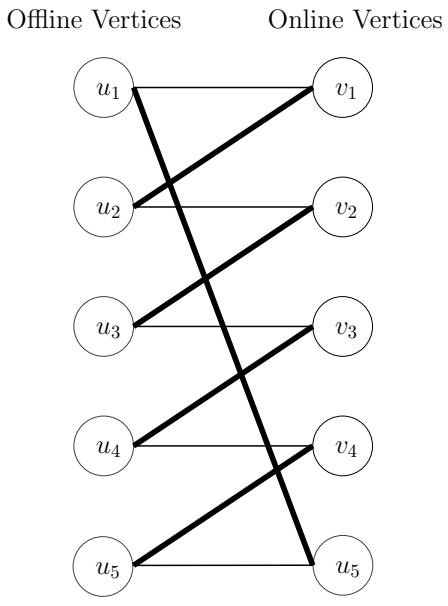
Chapter 4

Matching in $(2, 2)$-Regular Bipartite Graph – An Offline Optimal Algorithm

This thesis studies the performance of algorithms for online bipartite matching in graphs where the degree of vertices is 2. In other words, the bipartite graph $G(U, V, E)$ is $(2, 2)$ regular bipartite graph, and it has 2 perfect, disjoint matchings. For reference, see Figure 4.1.



(a) An instantiation of $(2, 2)$-regular bipartite graph on $n = 4$ vertices



(b) An instantiation of $(2, 2)$-regular bipartite graph on $n = 5$ vertices

Figure 4.1: Examples of $(2, 2)$-regular bipartite (or biregular) graphs on different number of vertices.

The online vertices are sampled from a Markov chain $\mathcal{M}$, imitating the nature of lazy random walks, whose state space is V, and the objective is to create a matching of maximum size of edges between U and V. We divide our work in 2 parts – first, we develop an optimal offline algorithm which attains the maximum possible matching in the final realization graph. Using this, we calculate the size of expected matching (under asymptotic conditions). The second part involves calculating the size of matching (again, under asymptotic conditions) for two significant and crucial algorithms in the field of online matching – Two Suggested Matching - Adaptive and Two Suggested Matching - Non Adaptive. Together with the previous results, we get a tight bound on asymptotic competitive ratio for these two algorithms.

In this chapter, we address the first part of the problem, and in Chapter 5, we carry out the analysis mentioned in the second part and derive the competitive ratios.

We begin this chapter by discussing the type graph and Markov chain on which we present our results, along with a brief overview of the optimal offline algorithm which we are designing. In the second half, we analyze this algorithm from algorithmic perspective and use the analysis to calculate the size of expected matching in offline scenario.

4.1 Problem Instance & Input Parameters

As stated in Section 2.4, for a problem instance of online bipartite matching in Markov chain model, we are given a type graph, $G(U, V, E)$ and a Markov chain $\mathcal{M}(\pi, M)$. We describe the type graph and Markov chain for our problem in the following subsection.

4.1.1 Type Graph

The type graph $G(U, V, E)$ is a bipartite graph, of size $2n$ with U and V being the offline and online vertex set respectively, and $|U| = |V| = n$. The vertex set is labeled with the set of natural numbers, $\mathbb{N}$. The edge set consists of the edges (u_i, v_i) for all $i \in [1, n]$. Additionally, for $i \in [1, n-1]$, the edges are (u_{i+1}, v_i), and for $i = n$, the edge is (u_1, v_n).

Two perfect and disjoint matchings exist in the graph structure, which we refer to as M_1 and M_2.

$$M_1 : \quad (u_i, v_i) \qquad \text{if } i \in [1, n],$$

$$M_2 : \begin{cases} (u_{i+1}, v_i) & \text{if } i \in [1, n-1], \\ (u_1, v_i) & \text{if } i = n. \end{cases}$$

We exploit the independence of these matchings to make decisions on which edge to include in the final matching.

4.1.2 Markov Chain

For the stochastic nature of input, we sample from a Markov chain based on the analysis of real world experimental data, which we report in Chapter 6. Markov chain $\mathcal{M}(\pi, M)$, with state space $S = V$ has the following properties. π – the initial distribution, taken to be uniform, i.e. the probability of each vertex in V being the first online request equals $\frac{1}{n}$, and M – the transition matrix, based on lazy random walk, has the probability of a vertex remaining in the same state or repeating itself being denoted by parameter p, and it transitions to other $n-1$ vertices with an equal probability of $\frac{1-p}{n-1}$. The transition probability is given using the following stochastic matrix:

$$M[i][i] = p \qquad \forall\, i \in [1, n]$$
$$M[i][j] = \frac{1-p}{n-1} \qquad \forall\, i, j \in [1, n], i \neq j$$

For example, for $p = \frac{1}{2}$, the transition matrix is:

$$M[i][i] = \frac{1}{2} \qquad \forall\, i \in [1, n]$$
$$M[i][j] = \frac{1}{2(n-1)} \qquad \forall\, i, j \in [1, n], i \neq j$$

An example of such Markov chain is shown in Figure 4.2.

4.1.3 Input Size

The input sequence, denoted using $\vec{\hat{v}}$ is a k length sequence of the form $(\hat{v_1}, \hat{v_2}, \ldots \hat{v_k})$. Each input item, $\hat{v_i}$ represents a type or is similar to a vertex in V. Note, the hat symbol '$\hat{}$' denotes that an

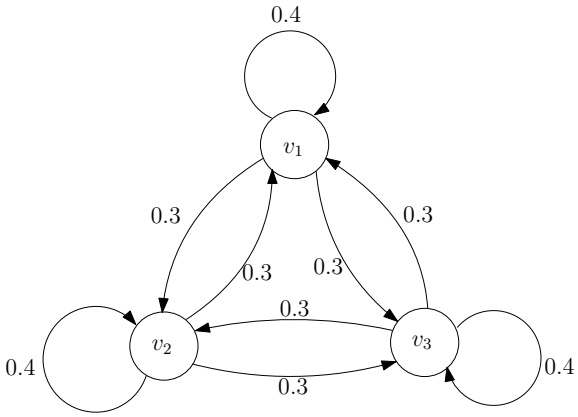


Figure 4.2: Example of a Markov chain on 3 states, with $p = 0.4$.

input item, an online vertex in the sequence and this is generally followed by denoting which $v \in V$ is the online vertex similar to. In contrast, if in this thesis we state that a vertex v_i (without the $\widehat{\ }$) has appeared in the sequence, we mean to say an online vertex having type v_i has appeared, as an shorthand expression. For online vertices, we have:

(1) $\widehat{v_1} \sim \pi$, and

(2) $\forall\, i \geq 2,\ \widehat{v_i} \sim M[\cdot \mid \widehat{v_{i-1}}]$,

where $M[\cdot \mid \widehat{v_{i-1}}]$ denotes the previous state of Markov chain is the vertex of 'type $\widehat{v_{i-1}}$', and the type of next sampled vertex, $\widehat{v_i}$ is only dependent on it.

For presenting our results, we represent the length of the sequence as a parameterized input, based on the size of the bi-partition of type graph. We denote the size of the input sequence, $k = an$, where n denotes the size of the offline (or the online) vertex set, and a represents the the ratio of input sequence and the size of either of the vertex sets. In the later sections, when we want to calculate the probability of certain input sequences, this comes into play by replacing an independent variable with a dependent variable and is better suited to present the results. We present an example as in Figure 4.3.

4.2 Offline Algorithm: Frequency Based Optimal Matching

In this section, we devise a new optimal offline algorithm, Frequency Based Optimal Matching and analyze its performance to compare with the two online algorithms. The offline optimal

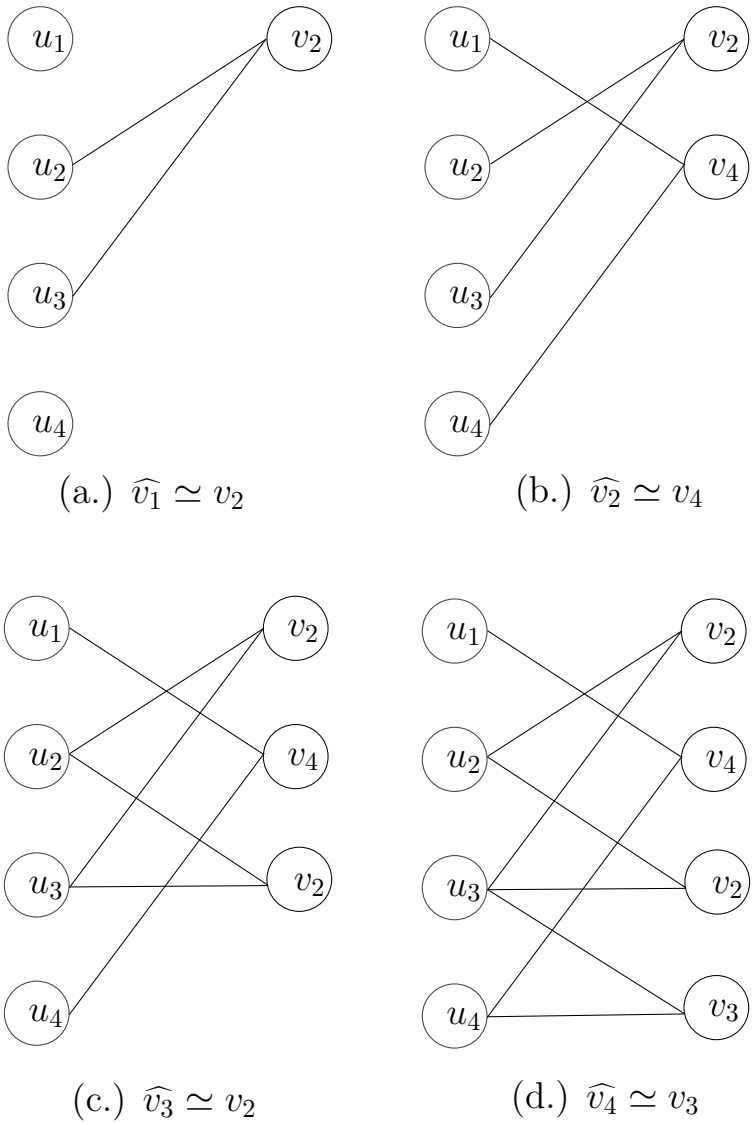


Figure 4.3: For type graph $(2,2)$-biregular graph of size $n = 4$, the following showcases the formation of the realization graph on input sequence $(\widehat{v_1}, \widehat{v_2}, \widehat{v_3}, \widehat{v_4})$, where the types of online vertices are $\widehat{v_1} \simeq v_2,\ \widehat{v_2} \simeq v_4,\ \widehat{v_3} \simeq v_2,\ \widehat{v_4} \simeq v_3$.

algorithm serves as a nice alternative to the Ford-Fulkerson and other augmenting path based algorithms, in such that analyzing its computation of total matching is relatively easier and it utilizes the independence of the two matchings in the type graph. The run time complexity of this algorithm is linear and dependent on the size of input sequence, $O(k)$.

An offline algorithm for an online problem can be interpreted as a scenario in which an algorithm starts making decisions after the entire input sequence has arrived. Hence in this problem, the

algorithm which we devise, Frequency Based Optimal Matching receives the final realization graph, composed of 2 bi-partitions, offline vertex set U and the set of online vertices having a type in V which have arrived online, and it finds the maximum matching for this graph. This algorithm is optimal and uses the total number of occurrences of each type of input vertex to decide the size of the matching. From here onward, we refer to this optimal algorithm as OPT. We now describe the algorithm functioning in brief.

Frequency Based Optimal Matching is an offline algorithm which uses the frequency of each online node $\widehat{v_i}$ to determine the number of matched offline vertices. An offline vertex, say u_i has neighbors v_i and v_{i+1}. After observing the complete input, if the frequency of vertices having online type $\widehat{v_i}$ and $\widehat{v}_{i+1}$ is 0, then u_i is certainly unmatched in the final matching, as either of its two online neighbors don't occur. Later in this chapter, we define precisely the conditions where an offline vertex remains unmatched, and the method to count the number of expected unmatched offline vertices.

We wish to calculate the expected number of matching achieved by OPT in asymptotic setting, or $\lim_{n\to\infty} \mathbb{E}[OPT]$. Before going to the working mechanism and calculations behind such, we define some structures that we use in our algorithm to calculate the size of matching.

4.2.1 Definitions

In this section, we define a few terms which will help us in understanding the working mechanism of the algorithm better and help in analyzing the algorithm to calculate the expected size of matching. We begin by defining the *frequency vector* of the sequence, which stores the total number of occurrences of each type of online vertex $v_i \in V$.

Definition. *The frequency vector, f for the input sequence, $\vec{\widehat{v}}$ is an array of length n, where $f[i]$ or f_i is the total number of occurrences of vertices which are similar to v_i in the input sequence. Note, if an online vertex $\widehat{v}$ has a type, or is similar to v_i ($\widehat{v} \simeq v_i$), then $N(\widehat{v}) = N(v_i)$.*

Sometimes, we refer to the phrase 'vertex of type v_i occurring in the input sequence' to 'vertex v_i occurring in the input sequence', when the context suffices. An example of the frequency vector, f for $n = 6$ is:

- $\vec{v} = v_1, v_2, v_2, v_5, v_4, v_2$
- $f = 1, 3, 0, 1, 1, 0$

Note that the indices of the frequency vector start with 1 and go up to n. Having defined the frequency vector, we can define a *block* as:

Definition. *A contiguous subsequence* $i,\ i+1, \ldots,\ i+\ell+1$ *(with wrap-around) is called a* block *of length* $\ell \geq 0$ *if the following conditions are satisfied:*

- $f_i = f_{i+\ell+1} = 0$,
- $f_j \neq 0 \ \forall\ j \in \{i+1, \ldots, i+\ell\}$.

We refer to indices i and $i+\ell+1$ as *end-points* of the block, and indices $j \in \{i+1, \ldots, i+\ell\}$ as *internal points* of the block. The length of a block is the number of internal points of the block, in this case ℓ. We don't include the endpoints in the size of the block, hence the size of possible blocks varies from 0 to $n-2$.

Continuing our above example, for $n = 6$, where the input sequence is $\overrightarrow{v} = (v_1, v_2, v_2, v_5, v_4, v_2)$, and the frequency vector is $f = [1, 3, 0, 1, 1, 0]$. Block B_1 starts at 3 and ends at 6, and Block B_2 starts at 6 and ends at 3.

Finally, we proceed to define a *miss block*, and a *complete block*.

Definition. *We say that a block B is a* miss *block if for every internal point j we have* $f_j = 1$. *Otherwise, if one of its internal points j satisfies* $f_j \geq 2$*, it is a complete block. By default, we say that a block B of length* $\ell = 0$ *is a miss block.*

Note that every block is either a miss block or a complete block, as these two cases are mutually exclusive and exhaustive. In the above example, we note that B_1 is a *miss block* of length 2, because $f_4 = f_5 = 1$, and B_2 is a *complete block* of length 2, because $f_2 \geq 2$.

4.2.2 Working Mechanism

Before we formally state the Frequency Based Optimal Matching algorithm, we discuss the idea behind calculating the matching in the realization graph using the count of miss blocks in the

frequency vector. We start with a basic observation that, given the size of offline vertex set, $|U| = n$, this is also the maximum possible size of matching for the problem. We now show that every occurrence of a *miss block* leads to exactly 1 unmatched offline vertex in OPT, hence decreasing the total number of possible matchings by 1, and there is no unmatched offline vertex corresponding to a complete block.

We start with describing a few basic cases of block size $\ell = 0, 1, 2, \ldots n-2$, and how an offline vertex remains unmatched because of these cases.

- $\ell = 0$: Suppose for an input sequence $\vec{\hat{v}}$, the frequency vector f incurs a miss block B of length 0 demarcated by endpoints i and $i+1$. This means in the input sequence, v_i and v_{i+1} never occurs. The corresponding common neighbor of these in U, u_i never gets matched, because neither of its 2 possible neighbors in the online set occurs. Hence, a block of length $\ell = 0$ leads to exactly 1 miss in the final matching.

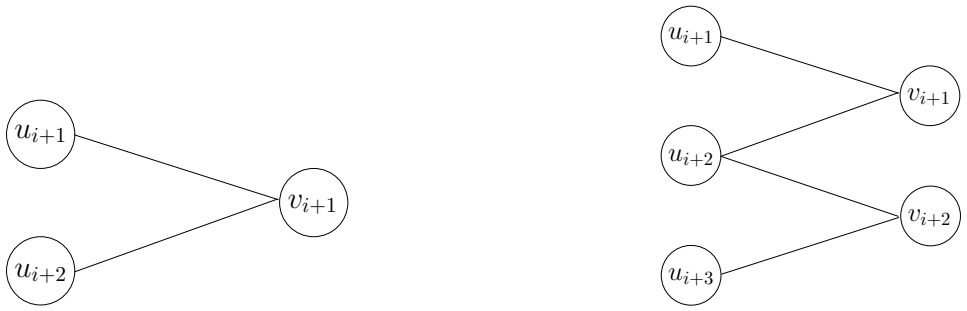
- $\ell = 1$: For a miss block B of length 1 with endpoints i and $i+2$, we have $f_i = 0,\ f_{i+1} = 1,\ f_{i+2} = 0$. This exhibits that in the input sequence, there was no occurrence of v_i and v_{i+2}, and only 1 occurrence of v_{i+1}. The neighbors of vertex v_{i+1} are u_{i+1} and u_{i+2}. The following 2 cases are possible:

  (1) OPT matches the vertex v_{i+1} with u_{i+1}, u_{i+2} remains unmatched in the final matching, because out of its 2 neighbors, v_{i+1} has occurred once and been already matched, and v_{i+2} never occurs.

  (2) A similar situation occurs, when OPT decided to match the vertex v_{i+1} with u_{i+2}. u_{i+1} remains unmatched in the final matching, because out of its 2 neighbors, v_{i+1} has occurred once and been already matched, and v_i never occurs.

  Hence, in any case, a block of length $\ell = 1$ leads to 1 miss in the final matching.

- $\ell = 2$: A miss block B of length 2, whose endpoints are denoted by i and $i+3$. $i+1$ and $i+2$ are the internal points which signify that there is exactly one occurrence of v_{i+1} and v_{i+2} type nodes in $\hat{v}$. Vertices of type v_i and v_{i+2} never appear in the input sequence. The following 3 cases of matching are possible:



(a) Example for $\ell = 1$ (b) Example for $\ell = 2$

Figure 4.4: Graph realizations depicting appropriate vertices for case $\ell = 1$ and $\ell = 2$.

(1) Match v_{i+1} to u_{i+1} and v_{i+2} to $u_{i+2} - u_{i+3}$ remains unmatched, because among its 2 neighbors, v_{i+2} is matched, and v_{i+3} never appears.

(2) Match v_{i+1} to u_{i+1} and v_{i+2} to $u_{i+3} - u_{i+2}$ remains unmatched, because among both of its neighbors, v_{i+1} and v_{i+2} are matched.

(3) Match v_{i+1} to u_{i+2} and v_{i+2} to $u_{i+3} - u_{i+1}$ remains unmatched, because among its 2 neighbors, v_{i+1} is matched, and v_i never appears.

Here too, a miss block of length $\ell = 2$ leads to 1 miss in the final matching.

- for an arbitrary $\ell \leq n - 2$: Let the input sequence $\hat{v}$ have an ℓ length miss block B. Denoting the endpoints of B with i and $i + \ell + 1$, and ℓ internal points as $i + 1, i + 1, \cdots, i + \ell$, we see that online vertices of type v_i and $v_{i+\ell+1}$ never appears. This leads to an imbalance of offline vertices to be matched and the online vertices in the miss block, with the former being always greater than the latter by one, and leads to exactly one missed offline vertex in the final matching by OPT. An example is depicted in Figure 4.5.

We also observe that any *complete block* would not lead to an unmatched online vertex. This can be proved by constructing a matching that leads to no unmatched vertices in the offline vertex set for the corresponding complete block. Assuming a complete block B of length ℓ, with endpoints i and $i + \ell + 1$, we have an internal point in the block g, such that for $g \in [i + 1, i + \ell]$, $f_i \geq 2$, by definition. Let g be the first such internal point of the block. We can construct a matching in this way:

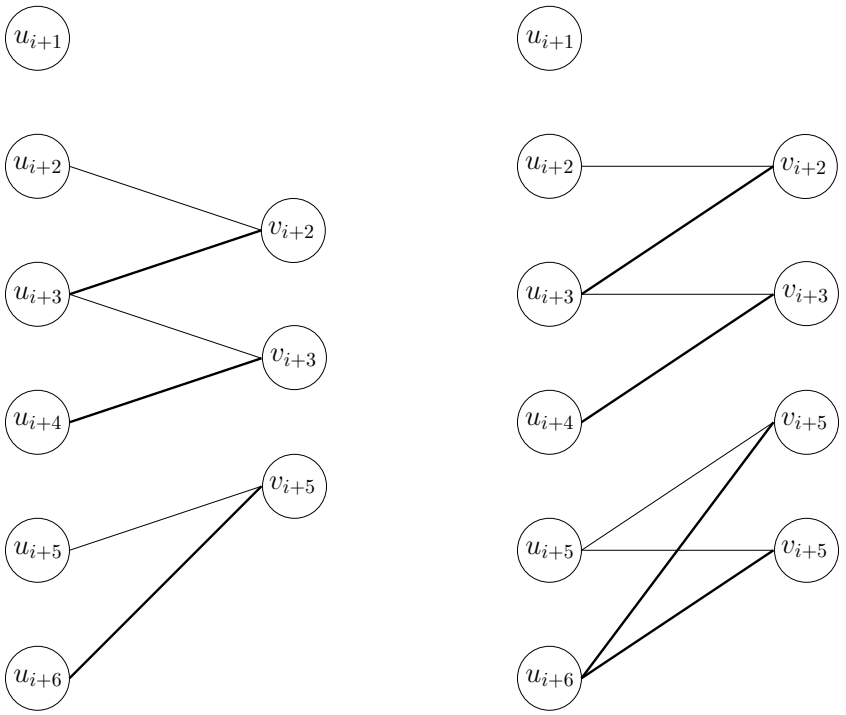


Figure 4.5: Two problem instances depicting miss blocks – (1) the first graph has 3 miss blocks from $(i+1, i+4)$, $(i+4, i+6)$ and $(i+6, i+1)$, whereas (2) the second graph has 2 miss blocks from $(i+1, i+4)$ and $(i+6, i+1)$. The block between $(i+4, i+6)$ is not a miss block, because v_{i+5} type vertex occurs twice. Matched offline vertices in the first graph are 3, 6 total vertices minus 3 miss blocks. Similarly for the second graph, matched offline vertices are 4.

- **Step 1:** For all vertices $v_{i+1}, v_{i+2}, \ldots, v_{i+g}$, we match them to $u_i, u_{i+1}, \ldots, u_{i+g-1}$. This leads to g matched vertices in the offline set.

- **Step 2:** For the remaining vertices, $v_{i+g}, v_{i+g+1}, \ldots, v_{i+\ell}$, we match them to $u_g, u_{g+1}, \ldots, u_{i+\ell}$. This is because we have an extra v_{i+g} available to match. This leads to $\ell - g + 1$ matched vertices in the offline vertex set.

Total number of matched offline vertices are $\ell + 1$. So none of the vertices remain unmatched for complete block B.

4.2.3 Algorithm Analysis

In the previous section, we showed that the occurrence of every miss block leads to an unmatched offline vertex. The size of the final matching in a bipartite graph is equal to the number of matched vertices on either side of the bi-partition, hence the matching here is equal to the total number of offline vertices minus the number of unmatched online vertices. $OPT(\vec{\hat{v}})$ or OPT is the size of the matching in the realization graph, which is equal to the number of matched offline vertices. We present the result in Lemma 2 as:

Lemma 2. *For an input sequence $\vec{\hat{v}}$, the size of the optimum matching in offline setting given by $OPT(\vec{\hat{v}})$, the number of matched offline vertices, which is equal to total number of offline vertices minus the number of occurrence of miss blocks in the frequency vector. Thus,*

$$OPT(\vec{\hat{v}}) = n - \textit{ number of miss blocks in frequency vector } f\,.$$

We will use this lemma to compute the expected value of OPT.

Equation (2) finds the size of matching for a particular input sequence which is randomly generated and the size varies for each sequence. Note that $\vec{\hat{v}}$ is random and dependent on the method which we employ to sample the vertices. We are not interested in the number of matched vertices for a 'particular' input sequence, rather we want to calculate the average or expected number of matched vertices. We redefine the above equation by introducing random variables, which help capture the randomness of the sequence and find the expected performance of OPT.

The key goal in calculating OPT is to find the expected number of *miss blocks* in the frequency vector for an input sequence. Observe that the miss blocks can be of different lengths, from 0 to $n-2$. Thus, we can state that the total count of miss blocks as the sum of number of miss blocks of each length from $\ell = 0$ to $n-2$.

To calculate the expected number of miss blocks of a particular length ℓ, we introduce an indicator random variable, $X_i[\ell]$ which indicates whether a miss block of length ℓ has occurred

starting at index i in the frequency vector, f.

$$X_i[\ell] = \begin{cases} 1 & \text{if a miss block of length } \ell \text{ occurs at position } i \\ 0 & \text{otherwise,} \end{cases}$$

where $i \in [1, n]$, and $\ell \in [0, n-2]$.

One can count the total number of miss blocks in the frequency vector in this way – count the total number of miss blocks of all possible lengths from $\ell = 0$ to $n-2$ for an index i and repeat this process for all indices from $i = 1$ to n (our declaration of frequency vector f start from 1). We can express this mathematically as, number of miss blocks of all possible length starting from a fixed position i is $\sum_{\ell=0}^{n-2} X_i[\ell]$. And then, the total number of miss blocks becomes the sum of miss blocks (of all possible lengths) starting from each index i in the frequency vector, where $i \in [1, n]$. Hence, the total number of miss blocks can be expressed as $\sum_{i=1}^{n} \sum_{\ell=0}^{n-2} X_i[\ell]$. Combining Lemma 2 and the above, Lemma 3 follows.

Lemma 3. *The size of matching obtained by an optimal offline algorithm over an input sequence $\vec{v}$, denoted by OPT is the difference between size of offline vertex set and the number of occurrence of miss blocks in the frequency vector. $X_i[\ell]$ indicates whether a miss block of length ℓ occurs at position i in the frequency vector, and the presence of a miss block corresponds to a singular unmatched offline vertex. The total number of miss blocks is expressed as the sum of indicator random variables $X_i[\ell]$ of all possible lengths, $\ell = 0$ to $n-2$, over all possible indices $i = 1$ to n in the frequency vector f.*

$$OPT = n - \sum_{i=1}^{n} \sum_{\ell=0}^{n-2} X_i[\ell] \tag{1}$$

Because we are interested in the expected size of matching, we take expectations on both sides of the equation, and using linearity of expectation and exchanging the order of summations concurrently, we obtain

$$\mathbb{E}[OPT] = n - \sum_{\ell=0}^{n-2} \sum_{i=1}^{n} \mathbb{E}[X_i[\ell]]$$

Note, by symmetry the expected number of an ℓ – length miss block starting at index $i = 1$

is equal to the expected number of ℓ – length miss block starting at index $i = 2$, and so on. The expected number of miss blocks is independent of their starting position in the frequency vector, as the Markov chain is symmetric around all vertices which can be generated, and the presence and absence of any vertex is symmetric. Therefore, $\mathbb{E}[X_1[\ell]] = \mathbb{E}[X_2[\ell]] = \cdots = \mathbb{E}[X_n[\ell]]$, and we replace all similar terms in the above equation with $\mathbb{E}[X_1[\ell]]$. Therefore, we obtain:

$$\begin{aligned}\mathbb{E}[OPT] &= n - \sum_{\ell=0}^{n-2}\sum_{i=1}^{n}\mathbb{E}[X_i[\ell]] \\ &= n - n\sum_{\ell=0}^{n-2}\mathbb{E}[X_1[\ell]]\end{aligned}$$

Normalizing the equation, and taking limits as we are interested in the asymptotic performance of the algorithm, we get

Lemma 4.

$$\lim_{n\to\infty}\frac{\mathbb{E}[OPT]}{n} = 1 - \lim_{n\to\infty}\sum_{\ell=0}^{n-2}\mathbb{E}[X_1[\ell]] \tag{2}$$

4.2.4 Calculating the Size of Expected Matching

Continuing from Lemma 4, we are interested in calculating the expected number of miss blocks of all possible lengths starting from a fixed index $i = 1$ in the frequency vector (for large values of n). As the expectation of an indicator random variable is equal to the probability of the corresponding event it indicates, it boils down here to calculating the probability of the 'associated event' – an input sequence which creates a miss block of length ℓ in the frequency vector at index $i = 1$.

In this section, we discuss how the above scenarios can be realized by what input sequences, and then we calculate the probability of such input sequences.

We first define $P(E_{\ell,n})$ to be the probability of an event in which an input sequence leads to the occurrence of a miss block of length ℓ in the frequency vector at index $i = 1$. As $\mathbb{E}[X_1[\ell]] = P(E_{\ell,n})$, we are interested in the value of $\lim_{n\to\infty}\sum_{\ell=0}^{n-2}P(E_{\ell,n})$.

Note, we can change the upper bound of the sum to infinity, because the probability of all events

where $\ell > n-2$ to be 0. We rewrite the above equation as:

$$\lim_{n\to\infty} \sum_{\ell=0}^{n-2} \mathbb{E}[X_1[\ell]] = \lim_{n\to\infty} \sum_{\ell=0}^{\infty} P(E_{\ell,n}) \tag{3}$$

In the above equation, we need to calculate the limit of of infinite sum. To calculate this, we make use of *Tannery's theorem*, which states the conditions under which it is possible to interchange the limits of an infinite summation to the sum of their limiting values. We first state the Tannery's theorem, and show that the conditions mentioned in the theorem are satisfied for the above equation.

Theorem 5 (Tannery's Theorem [69])**.** *Let* $S_n = \sum_{l=0}^{\infty} a_l(n)$ *and suppose that* $\lim_{n\to\infty} a_l(n) = b_l$. *If* $|a_l(n)| \leq M_l$ *and* $\sum_{l=0}^{\infty} M_l < \infty$, *then* $\lim_{n\to\infty} S_n = \sum_{l=0}^{\infty} b_l$.

To rephrase the above, the interchange of limits and summation of an infinite series of summands $(a_l(n))$ is possible if there exists a term (M_l) greater than the limiting value of summand (b_l), whose infinite sum converges. Applying Tannery's theorem to Eq 3, we establish that

Theorem 6.

$$\lim_{n\to\infty} \sum_{\ell=0}^{\infty} P(E_{\ell,n}) = \frac{e^{2(ap-a)}}{1-(e^{ap-a}\cdot a(1-p)^2)} \tag{4}$$

The rest of this chapter is dedicated to this proof, and using the results to calculate the expected size of matching for OPT. As there are many components involved in calculating this value, we present the proof of Theorem 6 using a 3 step framework with the help of Tannery's theorem, as stated below.

(1) First, we begin by calculating the probability of the event $E_{\ell,n}$, and calculate its limiting value. The input sequence associated with the event can be realized in many ways, and we calculate the probability of each such sequence.

(2) Next, we define an event $F_{\ell,n}$, such that the probability of this event is greater than the probability of the former event, $E_{\ell,n}$. Corresponding to the Tannery's theorem, this would be the required M_l such that $a_l(n) \leq M_l$. We then show the infinite sum of $P(F_{\ell,n})$ converges.

(3) The conditions of Tannery's theorem being satisfied, we now exchange the limits of sum of $P(E_{\ell,n})$ to be sum of the limiting values of $P(E_{\ell,n})$, and state Theorem 13.

Step 1 – Calculating probability of event $E_{\ell,n}$ and its limiting value

Our first step is to calculate the probability of event $E_{\ell,n}$, and find its limiting value. We elaborate upon the cases in which an input sequence leads to a miss block of length ℓ occurring in the frequency vector at index 1. We calculate the probability of different such input sequences, and the sum of its limiting values gives the limiting value of $P(E_{\ell,n})$.

We begin by discussing the basic case, for the occurrence of miss block of length $\ell = 0$, corresponding to the indicator random variable $X_1[0] = 1$. This happens in the input sequence realization when a node of type v_1 doesn't occur, and the following node v_2 also doesn't occur. The probability of this input sequence, denoted using $P(E_{\ell=0,n})$ is,

$$P(E_{\ell=0,n}) = \left(1 - \frac{2}{n}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1}$$

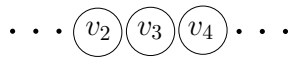
This follows by sampling from the previously defined input Markov chain $\mathcal{M}(\pi, M)$, parameterized with self remaining probability p. The first term denotes that the starting vertex cannot be v_1 and v_2, hence the probability of the first vertex can be uniformly any of the remaining $n-2$ of the n vertices. And all subsequent $an-1$ input items will not be transitioning into vertices v_1 and v_2, hence the probability of transition is being given by 1 minus $2 \times \frac{1-p}{(n-1)}$. The exponent $an-1$ denotes this transition probability for the remaining $an-1$ input terms.

Similarly for the occurrence of miss blocks of length $\ell = 1$, starting at index 1, in the input sequence, vertices of type, v_1, v_2, v_3, occur 0, 1 and 0 times respectively. The probability of such an input sequence would be

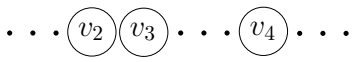
$$P(E_{\ell=1,n}) = \left(1 - \frac{3}{n}\right)\left(\frac{1-p}{n-1}\right)\left(1 - p - \frac{2(1-p)}{n-1}\right)\left(1 - \frac{3(1-p)}{n-1}\right)^{an-3}\binom{an-1}{1}$$

The $\binom{an-1}{1}$ expresses that the v_2 vertex can occur among $an-1$ positions in the input sequence. The $\frac{1-p}{n-1}$ denotes the incoming transition probability of v_2 and the term $1 - p - \frac{2(1-p)}{n-1}$ denotes the transition probability of just the next vertex after v_2 occurs in the input sequence. That vertex can be any of the n vertices except v_2 attributing to a negative p and the end points of the miss blocks vertices, v_1 and v_3, denoting a minus of $2 \times \frac{1-p}{n-1}$.

We also note that for all sequences such that $\ell \geq 2$, the vertices which serve as the internal point of the miss block and occur exactly once, can occur either *consecutively*, back to back in the input sequence, or occur *separately*. For example, as depicted in Fig. 4.6, for the $\ell = 3$ event, the vertices in the miss block which occur exactly once are v_2, v_3 and v_4. These vertices can occur either all separately, or any 2 of them can occur together and 1 separate from them or all 3 can occur together. Note, in all cases, the internal ordering of these vertices, whether v_2 occurs first or whether v_3 occurs first constitutes of different realizations of the input sequence, generally characterized by a factorial in the probability of the input sequences. And so on holds for all following events, for $\ell \geq 2$.



(a) All 3 vertices are together



(b) 2 out of 3 vertices are together

(c) None of the 3 vertices are together

Figure 4.6: Input sequences depicting 3 different ways vertices of type v_2, v_3 and v_4 can occur, where a miss block of length 3 occurs starting from index $i = 1$.

To capture these cases in which the vertices (which are the internal points of the miss block) appear altogether or separately in the input sequence, we rewrite the probability of the event $P(E_{\ell,n})$, as sum of two disjoint events $P(S_{\ell,n})$ and $P(T_{\ell,n})$.

$$\begin{aligned} P(E_{\ell,n}) &= P(S_{\ell,n}) + P(T_{\ell,n}) \quad \text{and given all limits exist,} \\ \lim_{n\to\infty} P(E_{\ell,n}) &= \lim_{n\to\infty} P(S_{\ell,n}) + \lim_{n\to\infty} P(T_{\ell,n}). \end{aligned} \tag{5}$$

Where $P(S_{\ell,n})$ denotes the probability of the events, in which none of the ℓ internal vertices of the miss block occur consecutively in the input sequence, and $P(T_{\ell,n})$ denotes the probability of the events in which at least two of the ℓ internal vertices of the miss block occur together in the input sequence. Note these events are mutually exclusive and exhaustive, i.e if an ℓ length miss blocks occurs, the input sequence has either none of the internal vertices as consecutive, or otherwise at least two of them occur together in the input sequence. We now proceed to calculate these two individual probabilities in the next section.

We begin by investigating the input sequences which lead to event $S_{\ell,n}$ – where a miss block of

length ℓ occurs and all the ℓ vertices are apart in the input sequence.

Such an input sequence can either begin with one of the ℓ internal vertices, or among the remaining $n - (\ell + 2)$ vertices which are not part of the miss block. This further divides the input sequence of events $S_{\ell,n}$ into 2 cases, $A_{\ell,n}$ and $B_{\ell,n}$ respectively.

$$\begin{aligned} P(S_{\ell,n}) &= P(A_{\ell,n}) + P(B_{\ell,n}), \text{ and given limits exists} \\ \lim_{n\to\infty} P(S_{\ell,n}) &= \lim_{n\to\infty} P(A_{\ell,n}) + \lim_{n\to\infty} P(B_{\ell,n}). \end{aligned} \tag{6}$$

We first calculate the probability of the input sequences of events $A_{\ell,n}$ and $B_{\ell,n}$, and then calculate their limiting values.

Let $A_{\ell,n}$ denote the event in which one of the ℓ vertices corresponding to the miss block happens to be the first vertex of the input sequence, and all ℓ vertices occur apart or non consecutively in the input sequence.

Claim 7.

$$\lim_{n\to\infty} P(A_{\ell,n}) = 0 \tag{7}$$

Proof. Observe that $P(A_{\ell,n})$ is equivalent to:

$$P(A_{\ell,n}) = \frac{\ell}{n} \cdot P(R)$$

where $\frac{\ell}{n}$ arises from the fact that ℓ out of n vertices can occur at the first position in the input sequence uniformly. And $P(R)$ is the probability of the remaining event of occurrence of an input sequence, in which all vertices corresponding to the internal endpoints of miss block of length ℓ occur exactly once, except the vertex which has already occurred at the first position. (And the two vertices which are the endpoints of the miss block do not occur.)

We can upper bound the $P(A_{\ell,n})$ by $\frac{\ell}{n}$, as the maximum value of $P(R)$ is 1. Therefore,

$$P(A_{\ell,n}) \leq \frac{\ell}{n}$$

Computing the limit, and lower bounding the value of probability $P(A_{\ell,n})$ by 0, we get

$$0 \leq \lim_{n\to\infty} P(A_{\ell,n}) \leq \lim_{n\to\infty} \left(\frac{\ell}{n}\right)$$

$$\lim_{n\to\infty} P(A_{\ell,n}) = 0$$

□

We repeat the same steps to calculate the other probability, $P(B_{\ell,n})$. We first calculate the probability of the input sequence associated with $B_{\ell,n}$, and then find its limiting value.

Let $B_{\ell,n}$ denote the event in which a miss block of length ℓ occurs in the frequency vector, such that none of the $\ell+2$ vertices corresponding to the miss block (which include the two external points) happens to be the first vertex of the input sequence.

Claim 8. *For all $\ell \geq 1$ we have:*

$$\lim_{n\to\infty} P(B_{\ell,n}) = a^{\ell} \cdot (1-p)^{2\ell} \cdot e^{-(\ell+2)(1-p)a} \tag{8}$$

Proof. Observe that we can compute $P(B_{\ell,n})$ as follows:

$$P(B_{\ell,n}) = \left(1 - \frac{\ell+2}{n}\right)\left(\frac{1-p}{n-1}\right)^{\ell}\left(1 - p - \frac{(\ell+1)(1-p)}{n-1}\right)^{\ell}\left(1 - \frac{(\ell+2)(1-p)}{n-1}\right)^{an-2\ell-1} \times$$
$$\times \left(\binom{an-\ell-1}{\ell} \cdot \ell!\right)$$

where the terms are explained as follows:

(1) $\left(1 - \frac{\ell+2}{n}\right)$: The first vertex of the input sequence cannot be any of the $\ell+2$ vertices in the miss block (and its endpoints), so the first in the sequence can be the remaining $n-(\ell+2)$ vertices uniformly.

(2) $\left(\frac{1-p}{n-1}\right)^{\ell}$: The ℓ vertices can occur anywhere except after themselves (because they occur only once). Therefore, the transition probability from any vertex to these vertices are $\frac{1-p}{n-1}$, and this transition happens ℓ times, for each vertex.

(3) $\left(1 - p - \frac{(\ell+1)(1-p)}{n-1}\right)^{\ell}$: The vertices just after the ℓ vertices of the miss block can not be (1) the

vertex themselves, or (2) any of the remaining $\ell - 1$ internal vertices or the 2 endpoints vertices of the miss block, hence any vertex except the $\ell + 1$ vertices. Therefore, the probability of next vertex can be anything except the sum of the probabilities of the above 2 events. The exponent of ℓ occurs because ℓ such transitions occur after each time one of the ℓ internal vertices occurs in the sequence.

(4) $\left(1 - \frac{(\ell+2)(1-p)}{n-1}\right)^{an-2\ell-1}$: The remaining $an - 2\ell - 1$ vertices can be any vertex except the $\ell + 2$ vertices of the miss block.

(5) $\binom{an-\ell-1}{\ell} \cdot \ell!$: The ℓ vertices can occur in the sequence in any permutation, hence $\ell!$ possible arrangements, and out of $an - 1$ positions in the input sequence, the ℓ vertices can occur at $\binom{an-\ell-1}{\ell}$ possible places under the condition that none of them occur together.

We can compute the limit by rearranging the internal terms and using the product limit laws as follows:

$$\lim_{n\to\infty} P(B_{\ell,n}) = \lim_{n\to\infty} \left\{ \left(1 - \frac{\ell+2}{n}\right) \left(\frac{1-p}{n-1}\right)^{\ell} \left(1 - p - \frac{(\ell+1)(1-p)}{n-1}\right)^{\ell} \times \right.$$
$$\left. \times \left(1 - \frac{(\ell+2)(1-p)}{n-1}\right)^{an-2\ell-1} \left(\binom{an-\ell-1}{\ell} \cdot \ell! \right) \right\}$$

$$\lim_{n\to\infty} P(B_{\ell,n}) = \lim_{n\to\infty} \left\{ \left(1 - \frac{\ell+2}{n}\right) \left(1 - p - \frac{(\ell+1)(1-p)}{n-1}\right)^{\ell} \left(1 - \frac{(\ell+2)(1-p)}{n-1}\right)^{an-2\ell-1} \right\} \times$$
$$\times \lim_{n\to\infty} \left\{ \left(\frac{1-p}{n-1}\right)^{\ell} \left(\binom{an-\ell-1}{\ell} \cdot \ell! \right) \right\}$$

The first limit can be computed as

$$\lim_{n\to\infty} \left\{ \left(1 - \frac{\ell+2}{n}\right) \left(1 - p - \frac{(\ell+1)(1-p)}{n-1}\right)^{\ell} \left(1 - \frac{(\ell+2)(1-p)}{n-1}\right)^{an-2\ell-1} \right\} =$$
$$= 1 \cdot (1-p)^{\ell} \lim_{n\to\infty} \left(1 - \frac{\ell+1}{n-1}\right)^{\ell} \cdot e^{a(\ell+2)(1-p)}$$
$$= (1-p)^{\ell} \cdot e^{a(\ell+2)(1-p)}$$

The second limit can be computed as

$$\begin{aligned}
\lim_{n\to\infty}\left\{\left(\frac{1-p}{n-1}\right)^{\ell}\left(\binom{an-\ell-1}{\ell}\cdot\ell!\right)\right\} &= (1-p)^{\ell}\lim_{n\to\infty}\left(\frac{(an-\ell-1)!}{(an-2\ell-1)!(n-1)^{\ell}}\right)\\
&= (1-p)^{\ell}\lim_{n\to\infty}\left(\sqrt{1+\frac{\ell}{an-2\ell-1}}\cdot\left(\frac{an-\ell-1}{n}\right)^{\ell}\right)\\
&= (1-p)^{\ell}\lim_{n\to\infty}\left(\left(a-\frac{\ell+1}{n}\right)^{\ell}\right)\\
&= (1-p)^{\ell}\cdot a^{\ell}
\end{aligned}$$

Therefore, the limiting value of $P(B)$ is:

$$\begin{aligned}
\lim_{n\to\infty} P(B) &= (1-p)^{\ell}\cdot e^{-a(\ell+2)(1-p)}\cdot(1-p)^{\ell}\cdot a^{\ell}\\
&= a^{\ell}\cdot(1-p)^{2\ell}\cdot e^{-a(\ell+2)(1-p)}
\end{aligned}$$

□

From Equations 6, 7 and 8, we get

$$\begin{aligned}
\lim_{n\to\infty} P(S_{\ell,n}) &= \lim_{n\to\infty} P(A_{\ell,n}) + \lim_{n\to\infty} P(B_{\ell,n})\\
&= a^{\ell}\cdot(1-p)^{2\ell}\cdot e^{-a(\ell+2)(1-p)} \qquad (9)
\end{aligned}$$

Up until now, we have calculated the limiting value of $P(S_{\ell,n})$ from Eq (5). We now calculate the probability of event $T_{\ell,n}$ in which at least 2 of the ℓ vertices of the miss block occur together. We prove that the probability of the sequences ascribed to this event is negligible, and its contributions to the final matching can be ignored.

We begin with an example for $\ell = 3$ or the event $T_{\ell=3,n}$. Let w_1, w_2, w_3 be the 3 internal vertices of the miss block. So event $T_{\ell=3,n}$ compromises of input sequences in which $(w_1, w_2), (w_2, w_3)\&(w_3, w_1)$ are pairwise together, and (w_1, w_2, w_3) are all together. The event in which w_1, w_2, w_3 are all together is covered under previous events in which any 2 of them are together. Using $C_{u,v}$ to describe an input sequence where vertices u and v are one of the internal

points of a miss block, and occur consecutively in the input sequence, we can express the above as:

$$T_{\ell=3,n} = C_{w_1,w_2} \cup C_{w_2,w_3} \cup C_{w_3,w_1}$$

Using union bound, the probability of the event $T_{\ell=3,n}$ can be upper bounded by the sum of probability of individual events C_{w_1,w_2}, C_{w_2,w_3} and C_{w_3,w_1}.

$$\begin{aligned} P(T_{\ell=3,n}) &= P\left(C_{w_1,w_2} \cup C_{w_2,w_3} \cup C_{w_3,w_1}\right) \\ &\leq P(C_{w_1,w_2}) + P(C_{w_2,w_3}) + P(C_{w_3,w_1}) \end{aligned}$$

Note, by symmetry $P(C_{w_1,w_2}) = P(C_{w_2,w_3}) = P(C_{w_3,w_1})$, and therefore

$$P(T_{\ell=3,n}) \leq 3P(C_{w_1,w_2})$$

We use logic similar to above to upper bound the probability of event $T_{\ell,n}$, where an arbitrary miss block of length ℓ occurs and prove its limiting value is 0.

Claim 9.

$$\lim_{n\to\infty} P(T_{\ell,n}) = 0 \tag{10}$$

Proof. $P(T_{\ell,n})$ is the probability of the event in which an ℓ size miss block occurs in the frequency vector, such that at least two of the ℓ vertices of the miss block occur together in the input sequence. Let's specify and name the nodes which occur consecutively as u and v.

We define $C_{u,v}$ to be an event, such that a miss block of length ℓ occurs in the frequency vector, and two vertices u and v from the miss block occur consecutively, in that order. Then for fixed ℓ length miss block, and every pair of u and v, we get –

$$T_{\ell,n} = \bigcup_{u,v \in L} C_{u,v}$$

where L denotes the set of internal vertices of the miss block. And using union bound

$$\begin{aligned} P(T_{\ell,n}) &= P\left(\bigcup_{u,v \in L} C_{u,v}\right) \\ &\leq \sum_{u,v \in L} P(C_{u,v}) \\ &\leq \binom{\ell}{2} P(C_{u,v}) \end{aligned} \tag{11}$$

where the last equality is followed by symmetry.

We now calculate bounds on probability of event $C_{u,v}$. Event $C_{u,v}$ has 2 vertices u and v occurring in the input sequence, let's say at positions i and $i+1$. The remaining vertices in the input sequence satisfy the other remaining conditions of $C_{u,v}$, that is the occurrence of an ℓ sized miss block from which two vertices have already occurred. Let the remaining event be R, and therefore

$$\begin{aligned} \underset{\text{fixed } i}{P(C_{u,v})} &= P(u \text{ occurs at position } i) \cdot P(v \text{ occurs at position } i+1) \cdot P(R) \\ &\leq P(u \text{ at } i) \cdot P(v \text{ at } i+1 \mid u \text{ at } i) \end{aligned}$$

And all possible positions of $i = 1$ to $an - 1$ in the input sequence, we have

$$\begin{aligned} P(C_{u,v}) &\leq \sum_{i=1}^{an-1} P(u \text{ at } i) \cdot P(v \text{ at } i+1 \mid u \text{ at } i) \\ &\leq \frac{1-p}{n-1} \sum_{i=1}^{an-1} P(u \text{ at } i) \\ &\leq \frac{1-p}{n-1} \sum_{i=1}^{an} P(u \text{ at } i) \end{aligned}$$

The probability of a node u from the set of online vertices, occurring at position i in the input sequence of length an can be calculated using Lemma 10.

Lemma 10.

$$P(\textit{vertex } v \textit{ at position } i) = \frac{1}{n}$$

Proof. Position i in the input sequence can be taken by any of the n online vertices having a type in

V. Note that

$$P(\text{position } i \text{ is occupied}) = P(\widehat{v_1} \text{ at } i) + P(\widehat{v_2} \text{ at } i) + \cdots P(\widehat{v_n} \text{ at } i]$$
$$1 = \sum_{j=1}^{n} P(\widehat{v_j} \text{ at } i)$$

Even though the probability of a node v occurring at position i is dependent on the node occurring at position $i-1$ because we are sampling from a Markov chain(except for position $i = 1$, in which case it is equivalent to $\frac{1}{n}$), in this case given no information about which node is present just before node u occurs at position i, the probability of all nodes are again equivalent. Therefore,

$$n \cdot P(v \text{ at } i) = 1$$

Hence, probability of a node v occurring at a particular position i in the input sequence is $\frac{1}{n}$ or $P(v \text{ at } i) = \frac{1}{n}$. □

Using this lemma, we get

$$P(C_{u,v}) \leq \frac{1-p}{n-1} \sum_{i=1}^{an} \frac{1}{n}$$
$$P(C_{u,v}) \leq \frac{1-p}{n-1} \cdot a \tag{12}$$

Combining Equations 11 and 12, we get

$$P(T_{\ell,n}) \leq \binom{\ell}{2} \cdot a \cdot \frac{1-p}{n-1}$$

Calculating its limits with respect to n tending to ∞, and lower bounding the probability by 0, we get

$$0 \leq \lim_{n\to\infty} P(T_{\ell,n}) \leq \lim_{n\to\infty} \binom{\ell}{2} \cdot a \cdot \frac{1-p}{n-1}$$
$$\lim_{n\to\infty} P(T_{\ell,n}) = 0$$

Hence, the contribution of event $T_{\ell,n}$ in asymptotic cases is 0. □

From Equation 9 and Equation 10, we conclude that

$$\lim_{n\to\infty} P(E_{\ell,n}) = \lim_{n\to\infty} P(S_{\ell,n}) + \lim_{n\to\infty} P(T_{\ell,n})$$
$$\lim_{n\to\infty} P(E_{\ell,n}) = a^{\ell} \cdot (1-p)^{2\ell} \cdot e^{-a(\ell+2)(1-p)} \tag{13}$$

Step 2 – Defining an upper bound on $P(E_{\ell,n})$ and proving the infinite summation converges

In the previous section, we calculated the limiting value of $P(E_{\ell,n})$. We now proceed with the 2^{nd} step of the process, defining an upper bound on the $P(E_{\ell,n})$, such that the infinite summation of the new term converges.

Corresponding to the ℓ length miss block in event $E_{\ell,n}$, we present an alternate strategy for calculating the associated probability. The event where this same ℓ length miss block occurs in the frequency vector and leads to the occurrence of these ℓ vertices exactly once in the input sequence can be bifurcated into these 2 cases – (1) the input sequence begins with one of these ℓ vertices, or (2) it begins with some vertex not in this set. We denote these two mutually exclusive and exhaustive events as $C_{\ell,n}$ and $D_{\ell,n}$ respectively.

Note, for both events $C_{\ell,n}$ and $D_{\ell,n}$ there can be four types of transitions among the vertices in the input sequence, based on predecessor-successor vertex combination. Let x be a vertex belonging to the set of ℓ vertices of miss block, and y be a vertex not in the miss block. Then, transitions in the input sequence can be of the following type:

(1) x to x: from a miss block vertex to a miss block vertex, the probability of transition is $\frac{1-p}{n-1}$

(2) y to x: from a non miss block vertex to a miss block vertex, again the probability of transition is $\frac{1-p}{n-1}$

(3) y to y: from a non miss block vertex to a miss block vertex, the probability of transition is $1-(\ell+2)\frac{1-p}{n-1}$

(4) x to y: from a miss block vertex to a non miss block vertex, the probability of transition is $1-p-(\ell+1)\frac{1-p}{n-1}$

In the input sequence when the ℓ vertices of miss block occur the total number of transitions into x, i.e. one of the vertices of the miss block is ℓ, and therefore the total transition probability is $\left(\frac{1-p}{n-1}\right)^{\ell}$. The remaining $an-\ell$ vertices are of the type y, one of the non miss block vertices. Such transitions can either occur from a miss block vertex (x) or from a non miss block vertex (y). Let t_1 and t_2 be the number of such transitions respectively, and $t_1+t_2 = an-\ell$. (There may be $an-\ell-1$ transitions, under the condition that the first input item doesn't get 'transitioned into', it is sampled uniformly.)

Then, the total transition probability leading *into* these y type vertices in the sequence would be $\left(1-p-\frac{(\ell+1)1-p}{n-1}\right)^{t_1}\left(1-\frac{(\ell+2)1-p}{n-1}\right)^{t_2}$. Note that, in asymptotic conditions, the transition probability from x to y is dominated by the transition probability from y to y.

$$1-p-(\ell+1)\frac{1-p}{n-1} \leq 1-(\ell+2)\frac{1-p}{n-1}$$
$$\frac{1-p}{n-1} \leq p \implies \frac{1-p}{p} \leq n-1 \implies \frac{1}{n} \leq p$$

as $p \in [0,1], n \in (1,\infty]$.

So in any input sequence, we can upper bound the transition probability of from any vertex to a y type vertex by the latter – $\left(1-\frac{(\ell+2)1-p}{n-1}\right)$. And the transition probability from any vertex to a x type vertex is $\left(\frac{1-p}{n-1}\right)$, as denoted earlier. Using these two facts, we compute an upper bound on the probability of events $C_{\ell,n}$ and $D_{\ell,n}$ as follows.

$$P(C_{\ell,n}) = \left(\frac{\ell}{n}\right)\left(\frac{1-p}{n-1}\right)^{\ell-1}\left(1-\frac{(\ell+2)(1-p)}{n-1}\right)^{an-\ell}\binom{an-1}{\ell-1}(\ell-1)!$$
$$P(D_{\ell,n}) = \left(1-\frac{\ell+2}{n}\right)\left(\frac{1-p}{n-1}\right)^{\ell}\left(1-\frac{(\ell+2)(1-p)}{n-1}\right)^{an-\ell-1}\binom{an}{\ell}(\ell!)$$

where the binomial coefficients arise from the placement of ℓ vertices in a an length input sequence and factorials arise from the internal arrangement of the ℓ vertices. Note in the earlier calculations, we had further divided the events into cases where ℓ vertices occur together or not, but we do not need to follow the same steps here. We want to remark that even though both of these probability calculations are for the same event $E_{\ell,n}$, the calculation of probability in the latter is more loosely upper bounded than the former, which helps us in deriving a corresponding M_l with respect to Tannery's theorem whose infinite summation can be proved to converge.

We upper bound the probability of event $E_{\ell,n}$ as follows:

$$\begin{aligned} P(E_{\ell,n}) &= P(C_{\ell,n}) + P(D_{\ell,n}) \\ &\leq \left(\frac{\ell}{n}\right)\left(\frac{1-p}{n-1}\right)^{\ell-1}\left(1-\frac{(\ell+2)(1-p)}{n-1}\right)^{an-\ell}\binom{an-1}{\ell-1}(\ell-1)! + \\ &+ \left(1-\frac{\ell+2}{n}\right)\left(\frac{1-p}{n-1}\right)^{\ell}\left(1-\frac{(\ell+2)(1-p)}{n-1}\right)^{an-\ell-1}\binom{an}{\ell}(\ell!) \end{aligned} \tag{14}$$

Note the R.H.S. of the equation corresponds to M_l of the Tannery's theorem. We now show that the infinite sum of the R.H.S. converges. We first introduce two lemmas, which help in simplifying the results.

Lemma 11.

$$\binom{n}{k} k! \leq n^k$$

Proof. Note,

$$n \cdot (n-1) \cdot (n-2) \ldots (n-k+1) \leq n^k \text{ and therefore}$$
$$\frac{n!}{(n-k)!} \leq n^k$$

Multiplying both sides by $\frac{1}{k!}$, we get

$$\binom{n}{k} \leq \frac{n^k}{k!} \implies \binom{n}{k} k! \leq n^k$$

Hence, $\binom{n}{k} k! \leq n^k$ □

Lemma 12. $\ln(x) - x \leq -1$

Proof. Note that $f(x) = \ln(x) - x$ is a decreasing function. For calculating the maxima of $f(x)$, we differentiate $f(x)$ and solve by equating its derivative to 0. Differentiating $f(x)$, we get

$$\frac{df(x)}{dx} = \frac{1}{x} - 1$$

And solving $\frac{1}{x} - 1 = 0$, we get $x = 1$. Therefore, the maximum value of $f(x)$ exists at $x = 1$, which

is

$$f(1) = \ln(1) - 1 = -1$$

Therefore, $f(x) = \ln(x) - x \leq 1$. □

Using these two lemmas, we further simplify the upper bounds on $P(C_{\ell,n})$ and $P(D_{\ell,n})$.

Upper bounding $P(C_{\ell,n})$

$$\begin{aligned}
P(C_{\ell,n}) &= \binom{\ell}{n}\binom{an-1}{\ell-1}(\ell-1)!\left(\frac{1-p}{n-1}\right)^{\ell-1}\left(1-\frac{(\ell+2)(1-p)}{n-1}\right)^{an-\ell} \\
&\leq 1\cdot(an)^{\ell-1}\cdot\left(\frac{1-p}{n-1}\right)^{\ell-1}\cdot e^{-\frac{(\ell+2)(1-p)(an-\ell)}{n-1}} \\
&\leq (a(1-p))^{\ell-1}\cdot\left(\frac{n}{n-1}\right)^{\ell-1}\cdot e^{-\frac{(\ell+2)(1-p)(an-a)}{n-1}}\cdot e^{-\frac{(\ell+2)(1-p)(a-\ell)}{n-1}} \\
&\leq (a(1-p))^{\ell-1}\cdot e^{\frac{\ell-1}{n-1}}\cdot e^{-\frac{(\ell-1)(1-p)(an-a)}{n-1}}\cdot e^{-\frac{3(1-p)(an-a)}{n-1}}\cdot e^{\frac{(\ell+2)(1-p)(\ell-a)}{n-1}} \\
&\leq \left\{(a(1-p))^{\ell-1}\cdot e^{-(\ell-1)(1-p)a}\right\}\cdot\left\{e\cdot e^{-3(1-p)a}\right\}\cdot e^{(\ell+2)(1-p)} \\
&\leq \left\{e^{(\ell-1)\ln(a(1-p))}\cdot e^{-(\ell-1)(1-p)a}\right\}\cdot\left\{e\cdot e^{-3(1-p)a}\right\}\cdot e^{3(1-p)}\cdot e^{(\ell-1)(1-p)} \\
&\leq e^{-(\ell-1)}\cdot\left\{e\cdot e^{-3(1-p)a}\cdot e^{3(1-p)}\right\}\cdot e^{(\ell-1)(1-p)} \\
&\leq e^{-(\ell-1)p}\cdot e^{3(1-p)(1-a)+1} \\
&\leq e^{-\ell p}\cdot e^{\{3(1-p)(1-a)+1+p\}} = e^{-\ell p}\cdot c_1
\end{aligned}$$

where c_1 is a constant.

And similarly upper bounding $P(D_{\ell,n})$,

$$
\begin{aligned}
P(D_{\ell,n}) &= \left(1 - \frac{\ell+2}{n}\right)\binom{an}{\ell}(\ell!)\left(\frac{1-p}{n-1}\right)^{\ell}\left(1 - \frac{(\ell+2)(1-p)}{n-1}\right)^{an-\ell-1} \\
&\leq 1 \cdot (an)^{\ell} \cdot \left(\frac{1-p}{n-1}\right)^{\ell} \cdot e^{-\frac{(\ell+2)(1-p)(an-\ell-1)}{n-1}} \\
&\leq (a(1-p))^{\ell} \cdot \left(\frac{n}{n-1}\right)^{\ell} \cdot e^{-\frac{(\ell+2)(1-p)(an-a)}{n-1}} \cdot e^{-\frac{(\ell+2)(1-p)(a-\ell-1)}{n-1}} \\
&\leq (a(1-p))^{\ell} \cdot e^{\frac{\ell}{n-1}} \cdot e^{-\frac{\ell(1-p)(an-a)}{n-1}} \cdot e^{-\frac{2(1-p)(an-a)}{n-1}} \cdot e^{\frac{(\ell+2)(1-p)(\ell+1-a)}{n-1}} \\
&\leq \left\{(a(1-p))^{\ell} \cdot e^{-\ell(1-p)a}\right\} \cdot \left\{e \cdot e^{-2(1-p)a}\right\} \cdot e^{(\ell+2)(1-p)} \\
&\leq \left\{e^{\ell \ln(a(1-p))} \cdot e^{-\ell(1-p)a}\right\} \cdot \left\{e \cdot e^{-2(1-p)a}\right\} \cdot e^{2(1-p)} \cdot e^{\ell(1-p)} \\
&\leq e^{-\ell} \cdot \left\{e \cdot e^{-2(1-p)a} \cdot e^{2(1-p)}\right\} \cdot e^{\ell(1-p)} \\
&\leq e^{-\ell p} \cdot e^{2(1-p)(1-a)+1} \\
&\leq e^{-\ell p} \cdot e^{\{2(1-p)(1-a)+1\}} = e^{-\ell p} \cdot c_2
\end{aligned}
$$

where c_2 is a constant.

Using the above bounds and substituting in Eq 14, we get

$$
\begin{aligned}
P(E_{\ell,n}) &\leq P(C_{\ell,n}) + P(D_{\ell,n}) \\
&\leq (c_1 + c_2)e^{-\ell p} = c \cdot e^{-\ell p}
\end{aligned}
$$

where c is a constant.

This R.H.S. provides us an M_l for completing the conditions of Tannery's theorem, and the infinite sum of $c \cdot e^{-lp}$, where $\ell \geq 0, p \geq 0$ converges to $c \cdot \frac{e^p}{e^p - 1}$. Hence, the conditions of Tannery's theorem are satisfied, and we can proceed to Step 3, exchanging the order of limits and summation.

Step 3 – Exchanging the order of limits and summation of $P(E_{\ell,n})$

We have shown that the conditions of Tannery's theorem are satisfied for $\lim_{n\to\infty} \sum_{\ell=0}^{n-2} P(E_{\ell,n})$ in Steps 1 and 2. We can now exchange the order of limits and sums, and using Eq 13, we get

$$\begin{aligned}
\lim_{n\to\infty} \sum_{\ell=0}^{n-2} P(E_{\ell,n}) &= \sum_{\ell=0}^{\infty} \lim_{n\to\infty} P(E_{\ell,n}) \\
&= \sum_{\ell=0}^{\infty} a^{\ell} \cdot (1-p)^{2\ell} \cdot e^{-(\ell+2)(1-p)a} \\
&= \sum_{\ell=0}^{\infty} \left(a(1-p)^2\right)^{\ell} \cdot e^{-\ell(1-p)a} \cdot e^{-2(1-p)a} \\
&= e^{-2(1-p)a} \cdot \sum_{\ell=0}^{\infty} \left(a(1-p)^2\, e^{-(1-p)a}\right)^{\ell} \\
&= e^{2(ap-a)} \sum_{\ell=0}^{\infty} (z \cdot e^{y})^{\ell}\ ,\ y = ap-a, z = a(1-p)^2 \\
&= \frac{e^{2(ap-a)}}{1-(e^{ap-a} \cdot a(1-p)^2)}
\end{aligned}$$

This concludes the proof of Theorem 6, and we obtain the result

$$\lim_{n\to\infty} \sum_{\ell=0}^{n-2} \mathbb{E}[X_1[\ell]] = \lim_{n\to\infty} \sum_{\ell=0}^{\infty} P(E_{\ell,n}) = \frac{e^{2y}}{1-(e^{y} \cdot a(1-p)^2)}\ ,\ y = ap-a$$

4.2.5 Size of Expected Matching

Using the above calculations, we can now calculate the size of expected matching constructed by an optimal offline algorithm in asymptotic conditions. We state the results in Theorem 13.

Theorem 13.

$$\lim_{n\to\infty} \frac{\mathbb{E}[OPT]}{n} = 1 - \frac{e^{2(ap-a)}}{1-(e^{ap-a} \cdot a(1-p)^2)} \tag{15}$$

This is the expected size of matching in a $(2,2)$-regular bipartite graph, where vertices are generated from a p parameterized lazy random walk based family of Markov chains by an offline optimal algorithm. We present a few corollaries next, for specific values of a and p.

Corollary 14. *For the conventional lazy random walk based Markov chain, where $p = \frac{1}{2}$, we obtain*

the expected size of matching as:

$$\lim_{n\to\infty} \frac{\mathbb{E}[OPT]}{n} = 1 - \frac{4}{4e^a - ae^{\frac{a}{2}}}$$

Corollary 15. *For the conventional lazy random walk Markov chain, where $p = \frac{1}{2}$, when the input sequence length is equal to the size of (a single) vertex set or $a = 1$, we get the expected size of matching as:*

$$\lim_{n\to\infty} \frac{\mathbb{E}[OPT]}{n} = 1 - \frac{4}{4e - \sqrt{e}} = 1 - 0.4336$$
$$\lim_{n\to\infty} \frac{\mathbb{E}[OPT]}{n} = 0.56636$$

We use these results to calculate the asymptotic competitive ratios for the family of Two Suggested Matching algorithms in the next chapter.

Chapter 5

Online Bipartite Matching in $(2, 2)$-Regular Bipartite Graph

In this chapter, we present an important family of online algorithms for bipartite matching, the Two Suggested Matching algorithms. The Two Suggested Matching - Non Adaptive algorithm (TSM-NA) was introduced by Feldman et al. in [26] and is of significant importance as it helped break the $1 - \frac{1}{e}$ barrier in online matching for the first time in nearly 20 years. This was further improved upon to the Two Suggested Matching Adaptive algorithm (TSM-A) in Manshadi et al. [62].

The power of Two Suggested Matching algorithms lies in obtaining 2 large disjoint matchings of the graph and using them to make a decision on whether to match the current vertex or not. Because our type graph can be easily decomposed into 2 separate disjoint matchings, we can easily perform the analysis of the online algorithm without worrying about specific ways of splitting the graph. We want to calculate the competitive ratio of these algorithms under the new settings and we present the analysis of the expected size of the matching obtained by these two online algorithms. We first present an overview of the algorithms, and depict a case where these algorithms result in different matching size for the same input sequence. Next, we show the intuition behind calculating the size of expected matching for these algorithms, and finish off their respective sections with calculations of the expected size of matching set.

5.1 Overview of Algorithms

Two Suggested Matching - Non Adaptive

Suppose the two disjoint matchings of the type graph are M_1 & M_2. In Two Suggested Matching - Non Adaptive, when a vertex arrives for the first time, it is matched to the offline vertex in accordance with the first matching M_1 if available, and when it arrives the second time, it is matched to the offline vertex in accordance to the second matching M_2, if available.

Two Suggested Matching - Adaptive

In the Two Suggested Matching - Adaptive, for each arrival of a vertex, it is first matched in accordance to matching M_1 if available. If not available, it is matched according to its neighbor in matching M_2, given it is available. $TSM-A$ adapts to the condition of availability of corresponding offline vertices when making a decision, whereas $TSM-NA$ makes the decision irrespective of that. For example in $TSM-NA$, the first arrival of a vertex of type v' will not be matched to its neighbor in $M_2(v')$ (if available) when its neighbor in $M_1(v')$ is not available, creating a missed chance of matching. An example is depicted in Figure 5.1.

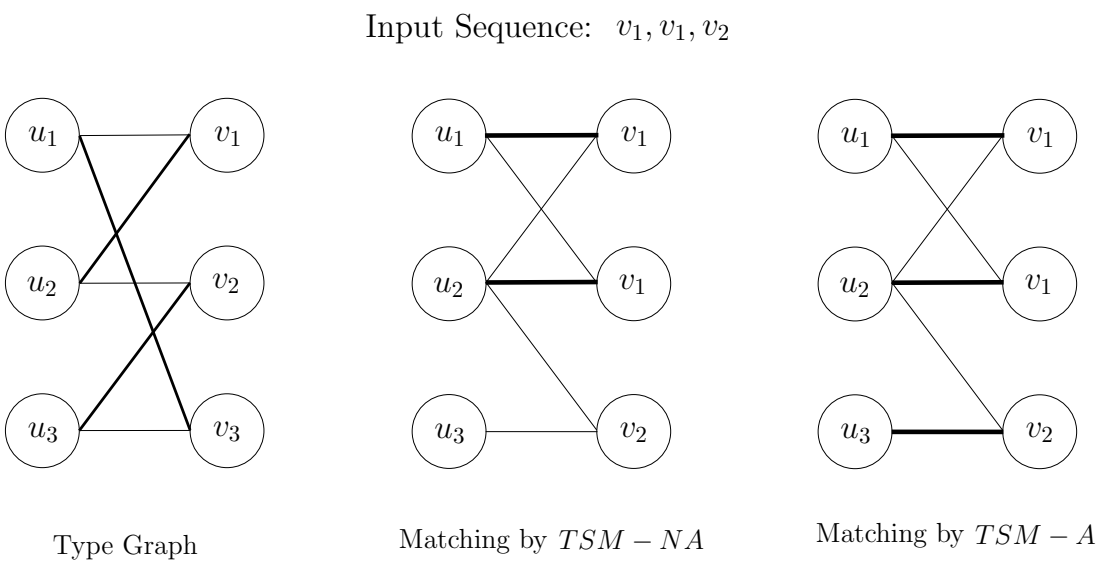


Figure 5.1: The first graph is the type graph, $(2,2)$-regular bipartite graph. The 2^{nd} and 3^{rd} graph depicts matching created by $TSM-NA$ and $TSM-A$ of size 2 and 3 respectively. The darker edges in the second and third graph depicts the edges in matching.

5.2 Two Suggested Matching - Non Adaptive

Recall for a bipartite type graph $G(U, V, E)$ and 2 disjoint perfect matchings, M_1 and M_2, the $TSM - NA$ algorithm matches the first occurrence of an online vertex with its neighbor in accordance to M_1, and for the second occurrence of the same online vertex, it matches in accordance to its neighbor in M_2 (given the offline vertices are available to match). We begin by discussing the scenarios in which an offline vertex gets matched. These scenarios give us an idea about how the input sequence should be for an offline vertex in U to get matched.

5.2.1 Working Mechanism

Let $u_i \in U$ be an offline vertex, which is matched in the final matching. For this requirement to be satisfied, the input sequence $\vec{\tilde{v}}$ must have any one of the three scenarios:

(1) A vertex of type v_i occurs at least once, given that no vertex of type v_{i-1} occurs before this. In this case, u_i gets matched with v_i.

(2) A vertex of type v_i occurs at least once, and exactly one vertex of type v_{i-1} occurs before this. In this case, u_i is available to be matched by v_i, because on the first occurrence of v_{i-1}, it gets matched according to M_1, to offline vertex u_{i-1} (given it is available). And u_i gets matched with v_i.

(3) There are two occurrence of vertex of type v_{i-1} before a vertex of type v_i occurs in the sequence. In this case, for the second occurrence of v_{i-1}, matching takes place according to M_2, and with vertex u_i.

Note for all 3 conditions, the indices of the vertices are wrapped around n, and start again from 1. In other words, for a pair of vertices v_i and v_{i+1} and a fixed n, such as $n = 5$, the vertex pair is v_3 and v_4 for $i = 3$ and v_5 and v_1 for $i = 5$. This holds for all similar discussions throughout the chapter.

These three scenarios cover all possible events in which u_i becomes matched in the final matching (or has an incident edge which is part of the matching set). Similar to OPT discussed in the previous chapter, we want to calculate the probability of the input sequences in these three scenarios, which helps in calculating the size of expected matching for the $TSM - NA$ algorithm.

5.2.2 Algorithm Analysis

Because the input sequence $\vec{\hat{v}}$ is a randomly generated, we want to find the expected expected number of matchings in the case of $TSM - NA$. To do so we make use of random variables once again which account for whether an offline vertex is matched in the final matching or not.

We define an indicator random variable X_i, which indicates whether $u_i \in U, i \in [1, n]$ is matched or not in the final matching.

$$X_i = \begin{cases} 1 & \text{if } u_i \text{ is matched in final matching} \\ 0 & \text{otherwise} \end{cases}$$

where $u_i \in U$.

The size of matching which is equal to the number of matched offline vertices can be calculated by iterating over and summing the sequence of indicator random variables, $(X_1, X_2, \cdots, X_n)$, as each indicator random variable being 1 indicates a matched offline vertex, and increments the sum by 1. We denote it using Lemma 16

Lemma 16. *The size of matching set obtained by $TSM - NA$ on an input sequence $\vec{\hat{v}}$, given by $ALG_{NA}(\vec{\hat{v}})$ is the sum over all indicator random variables, $(X_1, X_2, \cdots, X_n)$, where each X_i indicates whether an offline vertex is matched or not.*

$$ALG_{NA}(\vec{\hat{v}}) = \sum_{i=1}^{n} X_i \tag{16}$$

We use shorthand notation of referring $TSM_{NA}(\vec{\hat{v}})$ as TSM_{NA}. Taking expectation on both sides and using linearity of expectation, we get

$$\mathbb{E}[ALG_{NA}] = \sum_{i=1}^{n} \mathbb{E}[X_i]$$

As the type graph is symmetrical in terms of vertices and neighbor pairs, we see that $\mathbb{E}[X_1] = \mathbb{E}[X_2] = \cdots = \mathbb{E}[X_n]$. So, we replace the similar quantities with $\mathbb{E}[X_1]$ in the equations, and we

obtain

$$\mathbb{E}[ALG_{NA}] = n\mathbb{E}[X_1]$$

Normalizing and taking limits on both sides, we get

Lemma 17.

$$\lim_{n\to\infty} \frac{E[ALG_{NA}]}{n} = \lim_{n\to\infty} \mathbb{E}[X_1] \tag{17}$$

5.2.3 Calculating the Size of Expected Matching

From Lemma 17, we gather that we need to calculate the expected value of random variable X_1, which denotes whether u_1 is matched in the final matching or not. The vertex u_1 is matched if one of the 3 conditions mentioned in 5.2.1 are satisfied in the input sequence. Let E_n or (abbreviated to E where the context suffices) denote this event of u_1 being matched in the final matching, and E_1, E_2 and E_3 denote the 3 associated sub events. Then,

$$P(E) = P(E_1) + P(E_2) + P(E_3) \tag{18}$$

and, we rewrite the expected value of the indicator random variable X_1 as the probability of event E_n.

$$\lim_{n\to\infty} \mathbb{E}[X_1] = \lim_{n\to\infty} P(E) \tag{19}$$

Because we are interested in the limiting value of $\mathbb{E}[X_1]$, we calculate the limiting value of these probabilities, and get the following result:

Theorem 18.

$$\lim_{n\to\infty} P(E) = \frac{1+p}{2}\left(1 - e^{-2a(1-p)}\right) + \frac{1-p}{2}\left(1 - (1 + 2a(1-p)e^{-2a(1-p)})\right) \tag{20}$$

We derive the probabilities mentioned above in the next section.

CASE 1: Vertex of type v_1 occurs once, and no vertex of type v_n has occurred before it. This scenario can be realized by 2 possibilities:

(1) v_1 is the first vertex in the input sequence, with uniform probability of $\frac{1}{n}$, as any of the n vertices can be sampled uniformly. Probability of such an event is:

$$P(E_1) = \frac{1}{n} \text{ , and its limiting value is}$$
$$\lim_{n\to\infty} P(E_1) = 0$$

(2) v_1 is not the first vertex in the input sequence, and appears at q^{th} position in the input sequence for the first time. The transition probability from vertex at $q-1^{th}$ position to q^{th} position is given by $\frac{1-p}{n-1}$. And the first $an-q$ vertices have no restriction placed on them, other than they cannot be either v_1 or v_n. The probability of such an event, where vertex v_1 appears at a fixed q position is given by:

$$P(E_1)_{\text{fixed q}} = \left(1-\frac{2}{n}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{q-2}\left(\frac{1-p}{n-1}\right),\ q \in [2, an]$$

And the actual probability of the sub event (under this circumstance of $q \neq 1$) is given by summing the above probability for all possible values of q, and we obtain the following

$$P(E_1) = \sum_{q=2}^{an}\left\{\left(1-\frac{2}{n}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{q-2}\left(\frac{1-p}{n-1}\right)\right\}$$

The limiting value of this equation is:

$$\begin{aligned}
\lim_{n\to\infty} P(E_1) &= \lim_{n\to\infty}\left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)\sum_{q=0}^{an-2}\left\{\left(1-\frac{2(1-p)}{n-1}\right)^{q}\right\} \\
&= \lim_{n\to\infty}\left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)\left(\frac{1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-1}}{\frac{2(1-p)}{n-1}}\right) \\
&= 1\cdot\frac{1}{2}\cdot\left(1-e^{-2a(1-p)}\right) \\
&= \frac{1-e^{-2a(1-p)}}{2}
\end{aligned}$$

Combining the above 2 sub cases,

$$\lim_{n\to\infty} P(E_1) = \frac{1 - e^{-2a(1-p)}}{2}$$

CASE 2: Vertex of type v_1 occurs once, and only one vertex of type v_n has occurred before it. This scenario can be realized by the following four possibilities,

(1) v_n and v_1 are the first and second vertex of the sequence respectively. The probability of this sequence is:

$$P(E_2) = \frac{1}{n} \cdot \frac{1}{2(n-1)} \text{ , and its limiting value is}$$

$$\lim_{n\to\infty} P(E_2) = 0$$

(2) v_n is the first vertex of the sequence, and v_1 occurs at q^{th} position in the sequence ($q \neq 2$). The probability of such an input sequence, for a fixed value of q is:

$$\underset{\text{fixed q}}{P(E_2)} = \frac{1}{n}\left(1 - p - \frac{1-p}{n-1}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{q-3}\left(\frac{1-p}{n-1}\right),\ q \in [3, an]$$

Where the second term's probability is ascribed to the fact that the vertex just after v_n can not be v_n itself (deducting a value of p), and it can not be vertex v_1 (deducting a value of $\frac{1-p}{n-1}$). The rest of the terms have logic similar to their occurrence in Case 1. The probability of the input sequence in this sub case, for all values of q is given by:

$$P(E_2) = \sum_{q=3}^{an}\left\{\frac{1}{n}\left(1 - p - \frac{1-p}{n-1}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{q-3}\left(\frac{1-p}{n-1}\right)\right\}$$

The limiting value of the equation is:

$$\begin{aligned}
\lim_{n\to\infty} P(E_2) &= \lim_{n\to\infty} \frac{1}{n}\left(1-p-\frac{1-p}{n-1}\right)\left(\frac{1-p}{n-1}\right)\sum_{q=0}^{an-3}\left\{\left(1-\frac{2(1-p)}{n-1}\right)^{q}\right\} \\
&= \lim_{n\to\infty} \frac{1}{n}\left(1-p-\frac{1-p}{n-1}\right)\left(\frac{1-p}{n-1}\right)\left(\frac{1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}}{\frac{2(1-p)}{n-1}}\right) \\
&= \lim_{n\to\infty} \left(\frac{1-p}{2n}\right)\left(1-\frac{1}{n}\right)\left(1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}\right) \\
&= \left(\lim_{n\to\infty} \frac{1-p}{2n}\right)\cdot\left(\lim_{n\to\infty}\left\{\left(1-\frac{1}{n}\right)\left(1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}\right)\right\}\right) \\
&= 0\cdot\left(1-e^{-2a(1-p)}\right) \\
&= 0
\end{aligned}$$

(3) v_n and v_1 occurs consecutively at q^{th} and $q+1^{th}$ position in the input sequence, and $q \neq 1$. The probability of such an input sequence, for a fixed q is:

$$P(E_2)_{\text{fixed q}} = \left(1-\frac{2}{n}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{q-2}\left(\frac{1-p}{n-1}\right)^{2},\ q\in[2,an-1]$$

There would be $q-2$ transitions in the input sequence before the occurrence of v_n vertex in the sequence, and the two subsequent transitions of $\frac{1-p}{n-1}$ is for the occurrence of v_n and v_1 vertex respectively. The probability of the sequence for this sub case, for all values of q is given by

$$P(E_2) = \sum_{q=2}^{an-1}\left\{\left(1-\frac{2}{n}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{q-2}\left(\frac{1-p}{n-1}\right)^{2}\right\}$$

And the limiting value of the equation is:

$$
\begin{aligned}
\lim_{n\to\infty} P(E_2) &= \lim_{n\to\infty} \left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)^2 \sum_{q=0}^{an-3} \left\{\left(1-\frac{2(1-p)}{n-1}\right)^q\right\} \\
&= \lim_{n\to\infty} \left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)^2 \left(\frac{1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}}{\frac{2(1-p)}{n-1}}\right) \\
&= \lim_{n\to\infty} \left(\frac{1}{2}\right)\left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)\left(1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}\right) \\
&= \left(\lim_{n\to\infty} \frac{1-p}{n-1}\right) \cdot \left(\lim_{n\to\infty} \left\{\frac{1}{2}\left(1-\frac{2}{n}\right)\left(1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}\right)\right\}\right) \\
&= 0 \cdot \frac{\left(1-e^{-2a(1-p)}\right)}{2} \\
&= 0
\end{aligned}
$$

(4) v_n occurs at i^{th} position and v_1 occurs later at j^{th} position in the sequence, such that $i+2 \leq j$ and $i \neq 1$. The former condition specifies that these two vertices occurs non consecutively in the input sequence, and the latter specifies that the starting position isn't the first, which was already covered in above cases. Probability of such an input sequence, for fixed positions of i and j is:

$$
\begin{aligned}
\underset{\text{fixed } i,\, j}{P(E_2)} = &\left(1-\frac{2}{n}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{1-p}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{j-i-2} \times \\
&\times \left(\frac{1-p}{n-1}\right)(j-3)\,,\ i \in [2, an-2],\ j \in [4, an]
\end{aligned}
$$

The first three terms in the equation occur as follows: the occurrence of v_n at position i with probability $\frac{1-p}{n-1}$ is after $i-2$ transitions among vertices which are not v_n and v_1. The next term, $\left(1-p-\frac{1-p}{n-1}\right)$ is the transition probability from v_n to the next vertex, which can be any vertex except v_n and v_1. The next two terms follow a similar logic, but for vertex v_1, which occurs after $j-i-2$ transitions form the $q+1^{th}$ position. And a multiplicand of $j-3$ occurs because for a fixed position of v_n, v_1 can occur at $j-3$ positions in the input sequence (all j positions except $1^{st}, i^{th}$ and $i+1^{th}$). Hence, for a fixed position of v_n, $j-3$ such input sequences have the same

probability. The probability of the input sequence, over all valid values of i and j is given by:

$$P(E_2) = \sum_{j=4}^{an} \left\{ \left(1 - \frac{2}{n}\right) \left(\frac{1-p}{n-1}\right)^2 \left(1 - p - \frac{1-p}{n-1}\right) \left(1 - \frac{2(1-p)}{n-1}\right)^{j-4} (j-3) \right\}$$

And the limiting value of the equation is:

$$\lim_{n\to\infty} P(E_2) = \lim_{n\to\infty} \left(1 - \frac{2}{n}\right) \left(\frac{1-p}{n-1}\right)^2 \left(1 - p - \frac{1-p}{n-1}\right) \sum_{j=4}^{an} \left\{ \left(1 - \frac{2(1-p)}{n-1}\right)^{j} (j-3) \right\}$$

For calculating this sum, $\sum_{j=4}^{an} \left\{ \left(1 - \frac{2(1-p)}{n-1}\right)^{j} (j-3) \right\}$ we make use of the following observation. Let $\left(1 - \frac{2(1-p)}{n-1}\right) = f(n)$, abbreviated to as f. A geometric progression in f, between $\ell = 4$ to an can be simplified as:

$$\sum_{\ell=4}^{an} f^{\ell-3} = \sum_{\ell=1}^{an-3} f^{\ell} = \frac{f^{an-2} - f}{f-1}$$

Differentiating both sides with respect to f, we get the $L.H.S.$ as:

$$\frac{d}{df} \left(\sum_{l=4}^{an} f^{l-3} \right) = \sum_{l=4}^{an} f^{l-4} \cdot (l-3)$$

and the $R.H.S.$ (using quotient rule of differentiation) as:

$$\begin{aligned} \frac{d}{df} \left(\frac{f^{an-2} - f}{f-1} \right) &= \frac{\left((an-2) f^{an-3} - 1\right)(f-1) - 1\left(f^{an-2} - f\right)}{(f-1)^2} \\ &= \frac{f^{an-3}\left((an-3)f - (an-2)\right) + 1}{(f-1)^2} \end{aligned}$$

Substituting $f = 1 - \frac{2(1-p)}{n-1}$, we get the $L.H.S.$ as

$$\sum_{l=4}^{an} f^{l-4} \cdot (l-3) = \sum_{l=4}^{an} \left(1 - \frac{2(1-p)}{n-1}\right)^{l-4} (l-3)$$

and the $R.H.S.$ as

$$\frac{f^{an-3}\left((an-3)f-(an-2)\right)+1}{(f-1)^2} = \frac{\left(1-\frac{2(1-p)}{n-1}\right)^{an-3}\left((an-3)\left(1-\frac{2(1-p)}{n-1}\right)-an+2\right)+1}{\left(\frac{-2(1-p)}{n-1}\right)^2}$$
$$= \frac{\left(1-\frac{2(1-p)}{n-1}\right)^{an-3}\left((6-2an)\left(\frac{1-p}{n-1}\right)-1\right)+1}{4\left(\frac{1-p}{n-1}\right)^2}$$

Substituting the above result, we get

$$\lim_{n\to\infty} P(E) = \lim_{n\to\infty}\left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)^2\left(1-p-\frac{1-p}{n-1}\right)\sum_{j=4}^{an}\left\{\left(1-\frac{2(1-p)}{n-1}\right)^j(j-3)\right\}$$
$$= \lim_{n\to\infty}\left(1-\frac{2}{n}\right)\left(\frac{1}{4}\right)\left(1-p-\frac{1-p}{n-1}\right)\times$$
$$\times\left\{\left(1-\frac{2(1-p)}{n-1}\right)^{an-3}\left((6-2an)\frac{1-p}{n-1}-1\right)+1\right\}$$
$$= \frac{1-p}{4}\left(e^{-2(1-p)a}\left(-2a(1-p)-1\right)+1\right)$$
$$= \frac{(1-p)(1-(1+2a(1-p))e^{-2a(1-p)})}{4}$$

Combining the above 4 sub cases,

$$\lim_{n\to\infty} P(E_2) = \frac{(1-p)(1-(1+2a(1-p))e^{-2a(1-p)})}{4}$$

CASE 3: Vertex of type v_n occurs twice in the input sequence, given that vertex of type v_1 has not occurred before it. Again, this scenario can be realized by 4 possibilities, as follows:

(1) v_n occurs at the first and second position in the sequence. The probability of this sequence is:

$$P(E_3) = \frac{1}{n}\cdot\frac{1}{2(n-1)}\text{ , and its limiting value is}$$
$$\lim_{n\to\infty} P(E_3) = 0$$

(2) v_n is the first vertex of the input sequence, and the second v_n occurs at q^{th} position in the

sequence. The probability of such a sequence, for fixed q is

$$\underset{\text{fixed q}}{P(E_3)} = \frac{1}{n}\left(1 - p - \frac{1-p}{n-1}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{q-3}\left(\frac{1-p}{n-1}\right),\ q \in [3, an]$$

And the probability of the input sequence for all values of q is

$$P(E_3) = \sum_{q=3}^{an}\left\{\frac{1}{n}\left(1 - p - \frac{1-p}{n-1}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{q-3}\left(\frac{1-p}{n-1}\right)\right\}$$

This probability is same as the probability in Case 2.2, and hence we obtain the result,

$$\lim_{n\to\infty} P(E_3) = 0$$

(3) The first v_n occurs at q^{th} position in the sequence, and the second v_n occurs at $q + 1^{th}$ position in the sequence. The probability of such an input sequence for a fixed q is:

$$\underset{\text{fixed q}}{P(E_3)} = \left(1 - \frac{2}{n}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{q-2}\left(\frac{1-p}{n-1}\right)p,\ q \in [2, an-1]$$

The only difference in this probability to that of in Case 2.3 is that instead of two subsequent $\frac{1-p}{n-1}$ transitions, we now have a p transition denoting the consecutive occurrence of v_n vertices. The probability of the input sequence for all valid values of q is given by

$$P(E_3) = \sum_{q=2}^{an-1}\left\{\left(1 - \frac{2}{n}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{q-2}\left(\frac{1-p}{n-1}\right)p\right\}$$

The limiting value of the equation is:

$$\begin{aligned}
\lim_{n\to\infty} P(E_3) &= \lim_{n\to\infty} \left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right) p \sum_{q=0}^{an-3} \left\{\left(1-\frac{2(1-p)}{n-1}\right)^{q}\right\} \\
&= \lim_{n\to\infty} \left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right) p \left(\frac{1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}}{\frac{2(1-p)}{n-1}}\right) \\
&= \lim_{n\to\infty} \left(\frac{p}{2}\right)\left(1-\frac{2}{n}\right)\left(1-\left(1-\frac{2(1-p)}{n-1}\right)^{an-2}\right) \\
&= \frac{p}{2}\left(1-e^{-2a(1-p)}\right)
\end{aligned}$$

(4) The first v_n occurs at i^{th} position in the sequence, and the other v_n occurs at j^{th} position in the sequence, such that $i+2 \leq j$ and $i \neq 1$. This is similar to Case 2.4, where the constraints make sure that the two occurrence of vertices are not consecutive, and the first vertex in the input sequence is not v_n (as these cases are covered in the previous sub cases). The probability of such an input sequence for fixed i and j is:

$$\begin{aligned}
\underset{\text{fixed } i,\, j}{P(E_3)} = &\left(1-\frac{2}{n}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{1-p}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{j-i-2} \times \\
&\times \left(\frac{1-p}{n-1}\right)(j-3)\,,\ i \in [2, an-2], j \in [4, an]
\end{aligned}$$

For all valid values of i and j, the probability of the input sequence is given by

$$P(E_3) = \sum_{j=4}^{an} \left\{\left(1-\frac{2}{n}\right)\left(\frac{1-p}{n-1}\right)^{2}\left(1-p-\frac{1-p}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{j-4}(j-3)\right\}$$

This probability is same as the probability in Case 2.4, and we obtain the limiting value of the probability as

$$\lim_{n\to\infty} P(E_3) = \frac{(1-p)(1-(1+2a(1-p))e^{-2a(1-p)})}{4}$$

Combining the above 4 sub cases,

$$\lim_{n\to\infty} P(E_3) = \frac{p}{2}\left(1 - e^{-2a(1-p)}\right) + \frac{(1-p)(1-(1+2a(1-p))e^{-2a(1-p)})}{4}$$

We have now calculated the limiting value of probability of input sequences for all 3 sub events. The limiting value of probability for event E is the sum of these 3 limiting values and denotes the probability of occurrence of an input sequence in which vertex u_1 becomes matched in the final matching.

$$\begin{aligned}\lim_{n\to\infty} P(E) &= \frac{1-e^{-2a(1-p)}}{2} + \frac{p}{2}\left(1 - e^{-2a(1-p)}\right) + 2 \cdot \frac{(1-p)(1-(1+2a(1-p))e^{-2a(1-p)})}{4} \\ &= \frac{1+p}{2}\left(1 - e^{-2a(1-p)}\right) + \frac{1-p}{2}\left(1 - (1+2a(1-p))e^{-2a(1-p)}\right)\end{aligned}$$

This wraps up the proof of Theorem 18, and we have now obtained the results required to calculate the expected number of matchings by $TSM - NA$.

$$\lim_{n\to\infty} \mathbb{E}[X_1] = \lim_{n\to\infty} P(E) = \frac{1+p}{2}(1 - e^{-y}) + \frac{1-p}{2}(1 - e^{-y}(1+y)) \ , \ y = 2a(1-p)$$

5.2.4 Tight Bounds on Competitive Ratio

By computing the probability of the desired sequences above, now we can calculate the size of expected matching for the $TSM - NA$ algorithm. We express the result in Theorem 19 as follows.

Theorem 19.

$$\lim_{n\to\infty} \frac{E[ALG_{NA}]}{n} = \frac{1+p}{2}\left(1 - e^{-2a(1-p)}\right) + \frac{1-p}{2}\left(1 - (1+2a(1-p))e^{-2a(1-p)}\right) \tag{21}$$

Using Theorems 13 and 19, we can derive the asymptotic competitive ratio of Two Suggested Matching - Non Adaptive algorithm as:

$$\rho_{TSM-NA} = \frac{\frac{1+p}{2}\left(1 - e^{-2a(1-p)}\right) + \frac{1-p}{2}\left(1 - (1+2a(1-p))e^{-2a(1-p)}\right)}{1 - \frac{e^{2(ap-a)}}{1-(e^{ap-a}\cdot a(1-p)^2)}} \tag{22}$$

Once again, we present the results for some specific parameters of a and p.

Corollary 20. *For the conventional lazy random walk based Markov chain, where $p = \frac{1}{2}$, we obtain the expected size of matching as:*

$$\rho_{TSM-NA} = \frac{\frac{4(1-e^{-a})-ae^{-a}}{4}}{1-\frac{4}{4e^{a}-ae^{\frac{a}{2}}}}$$

Corollary 21. *For the conventional lazy random walk Markov chain, where $p = \frac{1}{2}$, when the input sequence length is equal to the size of (a single) vertex set or $a = 1$, we get the expected size of matching as:*

$$\rho_{TSM-NA} = \frac{1-\frac{5}{4e}}{1-\frac{4}{4e-\sqrt{e}}} = \frac{1-0.4598}{0.56636} = \frac{0.5401}{0.56636}$$

$$\rho_{TSM-NA} = 0.9536$$

Hence, under the lazy random walk Markov chain, $p = \frac{1}{2}$, Two Suggested Matching - Non Adaptive achieves a competitive ratio of $\rho_{TSM-NA} = 0.9536$.

Next, we analyze the adaptive version of the Two Suggested Matching algorithm.

5.3 Two Suggested Matching - Adaptive

Given a bipartite type graph $G(U, V, E)$ with 2 disjoint perfect matchings M_1 and M_2, for each input vertex of type $\widehat{v}$ in the sequence, the Two Suggested Matching - Adaptive ($TSM - A$) algorithm first checks its neighbor in accordance to $M_1(\widehat{v})$ to obtain a matching (if available). And it next checks it neighbor in accordance to $M_2(\widehat{v})$ to obtain a matching (if available). This is different from $TSM - NA$ in the sense that first occurrence of a vertex $\widehat{v}$ will only attempt to be matched to its neighbor in $M_1(\widehat{v})$, and if it is not available, it will not continue to look for its neighbor in $M_2(\widehat{v})$, creating a loss in its refusal to adapt.

5.3.1 Warm up: Case of $n = 3$

Similar to $TSM - NA$, we explain the scenarios in which an offline vertex remains unmatched in the final matching of the representation graph. We begin with an example case of $n = 3$ size type

graph which has 3 offline and online vertices. Focusing on an offline vertex u_1, we see it remains unmatched in the final matching because of either of these 2 conditions -

(1) No occurrence of v_1 till end of sequence (EoS) and no occurrence of v_3 till end of sequence, or

(2) No v_1 till EoS and one v_3 till EoS such that u_3 is unmatched when v_3 arrives.

The first condition makes sure there are no neighbors for u_1 to be matched to in the input sequence. And the second condition makes sure that the arrival of one of its neighbors, v_3 creates a matching with preference to M_1. For the second condition to be satisfied, it is needed that vertex u_3 remains unmatched until the occurrence of v_3. This is similar to our beginning problem statement, which deduces the cases where u_1 remains unmatched, illustrating the presence of a recursive structure in our problem.

The conditions of u_3 remaining unmatched when v_3 arrives for the first time in the input sequence can happen in the following situations:

(1) One v_3 till EoS and no occurrence of v_2 before v_3 arrives, or

(2) One v_3 till EoS and one v_2 before v_3 arrives such that u_2 is unmatched when v_2 arrives.

Substituting the above conditions 'in' the conditions for u_1 being unmatched in the final matching, we get:

(1) No v_1 till EoS and no v_3 till EoS, or

(2) No v_1 till EoS and one v_3 till EoS and no v_2 before v_3 arrives, or

(3) No v_1 till EoS and one v_3 till EoS and one v_2 before v_3 arrives, such that u_2 remains unmatched when v_2 arrives.

We see similar recursive conditions are set for vertex u_2 to be unmatched when v_2 arrives. We can state the conditions for this as follows:

(1) One v_2 till EoS and no occurrence of v_1 before v_1 arrive, or

(2) One v_2 till EoS and one v_1 before v_2 arrives such that u_1 remains unmatched when v_1 arrives.

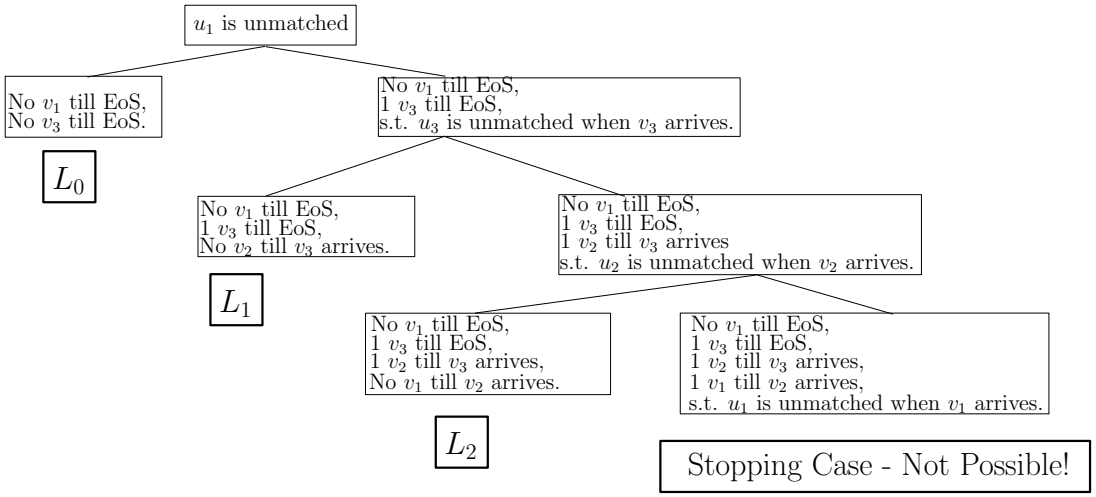


Figure 5.2: A recursive tree depicting the events where input sequences lead to u_1 remaining unmatched in the final matching for $n = 3$ size $(2, 2)$-regular bipartite graph.

Combining the above conditions, with the conditions of u_1 being unmatched in the final matching gives us:

(1) (L_0) No v_1 till EoS and no v_3 till EoS, or

(2) (L_1) No v_1 till EoS and one v_3 till EoS and no v_2 before v_3 arrives, or

(3) (L_2) No v_1 till EoS and one v_3 till EoS and one v_2 before v_3 arrives, and no v_1 before v_2 arrives, or

(4) No v_1 till EoS and one v_3 till EoS and one v_2 before v_3 arrives, and one v_1 before v_2 arrives, such that u_1 remains unmatched when v_1 arrives.

This final condition is contradictory because we have 2 complementing conditions conjugating, first that no v_1 arrives till EoS and the second that one v_1 is present in the sequence before v_2 arrives such that u_1 remains unmatched when v_1 arrived.

Since the last condition isn't possible, it also acts as a stopping case for the recursive tree, and we get 3 conditions for u_1 being unmatched in the final matching. We refer to the events corresponding to such conditions here as L_0, L_1 and L_2 respectively, for the leaf nodes at different levels of the recursive tree, as depicted in Figure 5.2.

5.3.2 Working Mechanism

We can generalize these conditions for the type graph of size n. Let $u_1 \in U$ be an offline vertex, which remains unmatched in the final matching. Note u_1 can be generalized to any arbitrary node $u_i \in U$, and the conditions will remain the same, just the indices would shift with respect to the difference between 1 and i. So, for u_1 to remain unmatched in the final realization graph, the input sequence $\vec{\hat{v}}$ must have any one of the n scenarios:

1. L_0 : No v_1 till EoS and no v_n till EoS, or
2. L_1 : No v_1 till EoS and one v_n till EoS and no v_{n-1} till EoS, or
3. L_2 : No v_1 till EoS and one v_n till EoS and one v_{n-1} before v_n arrives, and no v_{n-2} v_{n-1} before v_{n-1} arrives, or

   $\vdots$

n. L_{n-1} : No v_1 till EoS and one v_n till EoS and one v_{n-1} before v_n arrives, and no v_{n-2} v_{n-1} before v_{n-1} arrives, $\cdots$, one v_2 before v_3 arrives, and no v_1 before v_2 arrives.

These n conditions, L_0 to L_{n-1} represent the cases in which u_1 remains unmatched under the adaptive version, also shown in Figure 5.3. Using similar logic as before, we calculate the probability of such input sequences and use them to calculate the expected matching created under this algorithm.

5.3.3 Algorithm Analysis

Similar to the case of Frequency Based Optimal Matching and Two Suggested Matching - Non Adaptive, the input sequence $\vec{\hat{v}}$ is a randomly generated sequence, and we want to find the expected number of matchings in the case of $TSM-A$. Again, we define indicator random variables which denote whether an offline vertex is matched in the final matching or not.

We define an indicator random variable X_i, which indicates whether $u_i \in U, i \in [1, n]$ is unmatched or not in the final matching. X_i is the indicator for offline vertex u_i being unmatched in

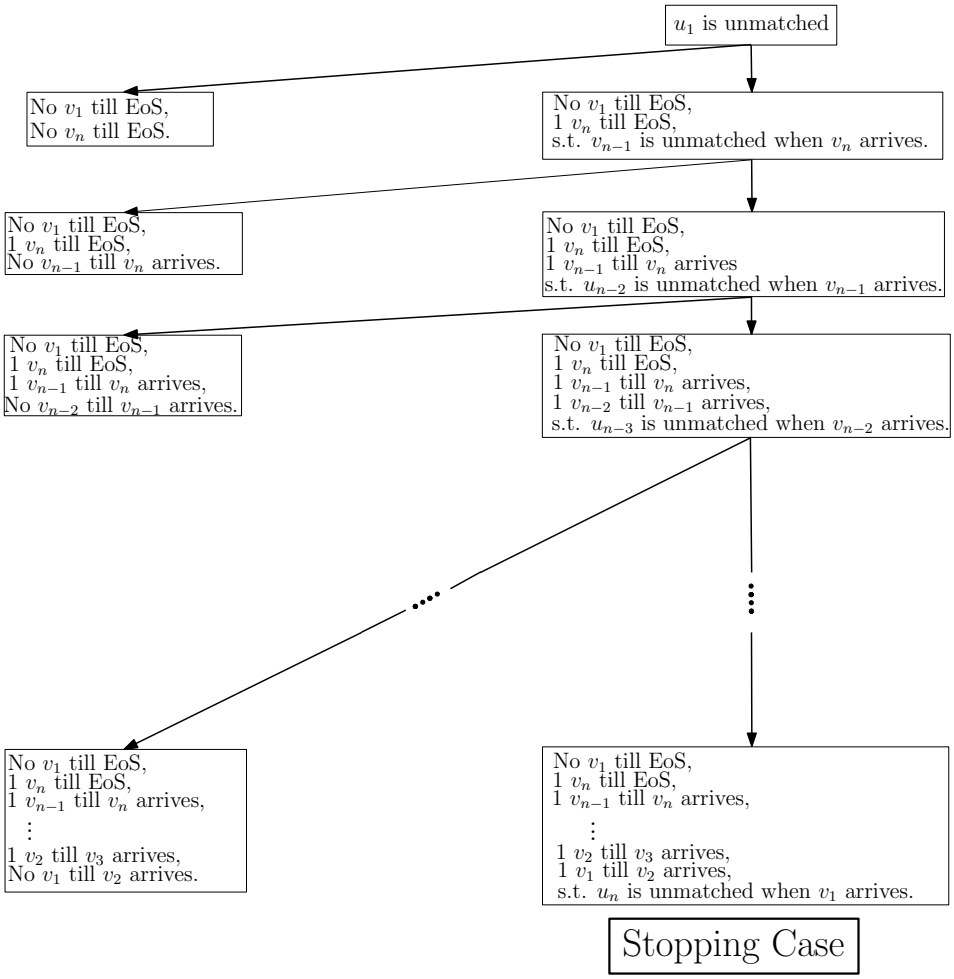


Figure 5.3: A recursive tree depicting the events where input sequences lead to u_1 remaining unmatched in the final matching.

the final matching, rather than being matched which was previously defined for $TSM - NA$.

$$X_i = \begin{cases} 1 & \text{if } u_i \text{ is unmatched in final matching} \\ 0 & \text{otherwise} \end{cases}$$

where $u_i \in U$. Note X_i here as the opposite meaning, of being *unmatched* as to when discussed in the previous section in the context of $TSM - NA$.

The total number of unmatched offline vertices can be calculated by iterating over and summing the sequence of indicator random variables, $(X_1, X_2, \cdots, X_n)$. Each matched offline vertex contributes to an increment in the sum. The total number of matched offline vertices is equal to the total

offline vertices, n minus the total unmatched offline vertices. We denote it using Lemma 22:

Lemma 22. *The size of matching set obtained under $TSM - A$ over an input sequence $\vec{\hat{v}}$, given by $ALG_{Ad}(\vec{\hat{v}})$ is the difference between total offline vertices and the sum over all indicator random variables, $(X_1, X_2, \cdots, X_n)$, where each X_i indicates whether an offline vertex is unmatched or not.*

$$ALG_{Ad}(\vec{\hat{v}}) = n - \sum_{i=1}^{n} X_i \tag{23}$$

Once again, we refer to $TSM_{Ad}(\vec{\hat{v}})$ as TSM_{Ad}. Taking expectation on both sides and using linearity of expectation, we obtain

$$\mathbb{E}[ALG_{Ad}] = n - \sum_{i=1}^{n} \mathbb{E}[X_i]$$

And once again, we use the symmetry of the type graph structure and see that $\mathbb{E}[X_1] = \mathbb{E}[X_2] = \cdots = \mathbb{E}[X_n]$. Replacing all equivalent terms in the expression with $\mathbb{E}[X_1]$, we get

$$\mathbb{E}[ALG_{Ad}] = n - n \cdot \mathbb{E}[X_1]$$

Finally, we normalize the value with respect to n, and take limits on both sides, we obtain

Lemma 23.

$$\lim_{n\to\infty} \frac{\mathbb{E}[ALG_{Ad}]}{n} = 1 - \lim_{n\to\infty} \mathbb{E}[X_1] \tag{24}$$

5.3.4 Calculating the Size of Expected Matching

Carrying on from Lemma-23, we see the need to calculate the expected value of random variable X_1, which denotes whether u_1 is unmatched in the final matching or not. The vertex u_1 is matched if one of the n conditions mentioned in 5.3.2 is satisfied in the input sequence. Because X_1 is a random variable, the expected value of X_1 is equal to the sum of probability of events corresponding to when

$X_1 = 1$, and therefore

$$\mathbb{E}[X_1] = \sum_{k=0}^{n-1} P(L_k)$$

$$\lim_{n\to\infty} \mathbb{E}[X_1] = \lim_{n\to\infty} \sum_{k=0}^{n-1} P(L_k) \tag{25}$$

Once again, we make use of Tannery's theorem and find conditions to switch the limit of sums of $P(L_k)$ to sum of limits. We define a step by step framework to computer the value in Theorem 24.

Theorem 24.

$$\lim_{n\to\infty} \sum_{k=0}^{\infty} P(L_k) = e^{(1-p)(e^{-a(1-p)}-2a)} \tag{26}$$

Note that we can change the upper limit of summation from $n-1$ to ∞, by simply observing that probability of events L_k, $k \geq n$ is 0, as they are impossible to happen. We now present the framework to calculate this value.

(1) The first step is calculating the limiting value of $P(L_k)$. Starting with computing the probability of the first few leaf nodes for $k = 0, 1, 2$, we then calculate the value of $P(L_k)$ for an arbitrary level k. The value of $P(L_k)$ is quite complex and large expression, which we simplify further by introducing *Gaussian binomial coefficients*. We introduce expressions upper and lower bounding the value of $P(L_k)$, and calculate its limits which coincide, and in turn gives us the limiting value of $P(L_k)$. Let this term, the value of $P(L_k)$ be known as a_l.

(2) Next, we define an M_k corresponding to $P(L_k)$, such that $P(L_k) \leq M_k$ and the infinite sum of M_k, from $k = 0$ to ∞ converges.

(3) With the conditions of Tannery's theorem satisfied, we finally calculate the infinite sum of a_l (the limiting value) from 0 to ∞, which gives us the required L.H.S. $\lim_{n\to\infty} \sum_{k=0}^{n-1} P(L_k)$.

Before beginning with the calculation of the limiting values of $P(L_k)$, we define two lemmas that are useful in calculating the probabilities in the first step.

Lemma 25. *The limiting value of the probability of an event E_n, in which nodes of type v_i and v_j appear in the input sequence consecutively is 0.*

Proof. Let E_n be an event in which nodes of type v_i and v_j appear in the input sequence consecutively, or v_i, v_j is a subsequence present in the input sequence. Then

$$P(E_n) \leq \frac{1-p}{n-1}$$

And the limiting value of probability of such a sequence for large n is

$$0 \leq \lim_{n\to\infty} P(E_n) \leq \lim_{n\to\infty} \frac{1-p}{n-1}$$
$$\lim_{n\to\infty} P(E_n) = 0$$

And every event $G_n \subseteq E_n$ has $P(G_n) \leq P(E_n)$, therefore using *sandwich theorem* once again, we get its limiting value, $\lim_{n\to\infty} P(G_n) = 0$. □

Lemma 26. *The limiting value of the probability of an event E_n, in which node v_i appears at a fixed position i in the input sequence is 0.*

Proof. Let E_n be an event, in which node v_i appears at a fixed position i in the input sequence. The probability of this event is

$$P(E_n) = P(v_i \text{ at position } i) = \frac{1}{n} \qquad \text{(Lemma 10)}$$

The limiting value of probability of such an event is

$$\lim_{n\to\infty} P(E_n) = 0$$

And any subsequent events $G_n \subseteq E_n$ follows the same, $\lim_{n\to\infty} P(G_n) = 0$. □

We now proceed with the 3 step framework to calculate the results in Theorem 24.

Step 1: Limiting value of $P(L_k)$

We start by calculating the probability of input sequences for a few initial leaves $L_0, L_1, L_2, \ldots$ which satisfy the conditions of $X_1 = 1$, or u_1 being unmatched.

(1) L_0: The condition for u_1 remaining unmatched in leaf L_0 is no occurrence of v_1 till EoS and no occurrence of v_n till EoS. The probability of such an input sequence and its limiting value is:

$$P(L_0) = \left(1 - \frac{2}{n}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1}$$

$$\lim_{n\to\infty} P(L_0) = e^{-2(1-p)a}$$

(2) L_1: The conditions for L_1 leaf is no v_1 till EoS, one v_n till EoS and no v_{n-1} till v_n arrives. The input sequence looks similar to – This can be realized in 3 ways:

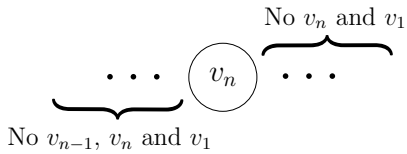


Figure 5.4: A visualization of input sequence for L_1.

2.1 v_n is the 1^{st} term of the sequence. The probability of such a sequence is:

$$P(L_1) = \frac{1}{n}\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1}$$

$$\lim_{n\to\infty} P(L_1) = 0$$

2.2 v_n is the last term of the sequence. The probability of such a sequence is:

$$P(L_1) = \left(1 - \frac{3}{n}\right)\left(1 - \frac{3(1-p)}{n-1}\right)^{an-2}\left(\frac{1-p}{n-1}\right)$$

$$\lim_{n\to\infty} P(L_1) = 0$$

2.3 v_n occurs at ℓ^{th} position, after $\ell - 1$ transitions. The probability of a single such sequence, for a fixed value of ℓ, $\underset{\ell \text{ fixed}}{P(L_1)}$ is given by

$$\left(1 - \frac{3}{n}\right)\left(1 - \frac{3(1-p)}{n-1}\right)^{\ell-2}\left(\frac{1-p}{n-1}\right)\left(1 - p - \frac{1-p}{n-1}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{an-\ell-1}$$

where ℓ varies from 2 to $an - 1$. The terms arising in the expression have a similar logic to

their appearance, as discussed in the probability calculations of $FBOM$ and $TSM - NA$. The total probability is the sum of the above probability for an individual ℓ, for all possible values of $\ell = 2$ to $an - 1$. Hence,

$$P(L_1) = \sum_{\ell=2}^{an-1} \left\{ \left(1 - \frac{3}{n}\right) \left(1 - \frac{3(1-p)}{n-1}\right)^{\ell-2} \left(\frac{1-p}{n-1}\right) \left(1 - p - \frac{1-p}{n-1}\right) \times \right.$$
$$\left. \times \left(1 - \frac{2(1-p)}{n-1}\right)^{an-\ell-1} \right\}$$

Grouping the terms independent of the index of summation ℓ separately, we can modify the expression as

$$P(L_1) = \left(1 - \frac{3}{n}\right) \left(\frac{1-p}{n-1}\right) \left(1 - p - \frac{1-p}{n-1}\right) \left(1 - \frac{3(1-p)}{n-1}\right)^{-2} \times$$
$$\times \left(1 - \frac{2(1-p)}{n-1}\right)^{an-1} z_1,$$

where

$$z_1 = \sum_{\ell=2}^{an-1} \left\{ \left(1 - \frac{3(1-p)}{n-1}\right)^{\ell} \left(1 - \frac{2(1-p)}{n-1}\right)^{-\ell} \right\} = \sum_{\ell=2}^{an-1} \left(\frac{n+3p-4}{n+2p-3}\right)^{\ell}$$
$$= \sum_{\ell=2}^{an-1} \left(1 - \frac{1-p}{n+2p-3}\right)^{\ell}.$$

(3) L_2: The conditions for L_2 is no v_1 till EoS, one v_n till EoS, one v_{n-1} till v_n arrives and no v_{n-2} till v_{n-1} arrives. The input sequence looks like

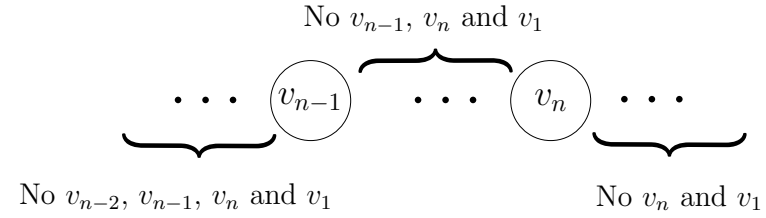


Figure 5.5: A visualization of input sequence for L_2.

These conditions can be realized in several different ways depending on the relative positions of v_{n-1} and v_n. Listing out the conditions based on the relative position of v_{n-1} and v_n we get,

3.1 v_{n-1} at first, v_n at second position:

3.2 v_{n-1} at first, v_n i^{th} at position

3.3 v_{n-1} at first, v_n at last position

3.4 v_{n-1} at i^{th} position, v_n $i+1^{th}$ at position

3.5 v_{n-1} at i^{th} position, v_n at last position

3.6 v_{n-1} at previous to last position, v_n at last position

For events concerning input sequences under 3.1, 3.4, and 3.6, the limiting value of the probability of the sequence is 0 by Lemma 25. The same follows for events concerning input sequences under 3.2, 3.3, and 3.5 by Lemma 26. The contributing sequence where v_{n-1} and v_n both have unanchored positions possible in the sequence.

3.7 v_{n-1} at i^{th} position, v_n at j^{th} position: The probability of such a sequence, for fixed i and j is:

$$\begin{aligned}\underset{i,j \text{ fixed}}{P(L_2)} = &\left(1-\frac{4}{n}\right)\left(1-\frac{4(1-p)}{n-1}\right)^{i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{2(1-p)}{n-1}\right)\times \\ &\times\left(1-\frac{3(1-p)}{n-1}\right)^{j-i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{1-p}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{an-j-1}\end{aligned}$$

where i varies from 2 to $an-3$ and j varies from 4 to $an-1$. The total probability of all input sequences under L_2 in case 3.7 is given by summing over all possible values of i and j which is:

$$\begin{aligned}P(L_2) = &\sum_{i=2}^{an-3}\sum_{j=4}^{an-1}\left\{\left(1-\frac{4}{n}\right)\left(1-\frac{4(1-p)}{n-1}\right)^{i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{2(1-p)}{n-1}\right)\times\right. \\ &\left.\times\left(1-\frac{3(1-p)}{n-1}\right)^{j-i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{1-p}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{an-j-1}\right\} \\ = &\left(1-\frac{4}{n}\right)\left(1-\frac{4(1-p)}{n-1}\right)^{-2}\left(\frac{1-p}{n-1}\right)^{2}\left(1-p-\frac{2(1-p)}{n-1}\right)\left(1-\frac{3(1-p)}{n-1}\right)^{-2}\times \\ &\times\left(1-p-\frac{1-p}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{an-1}\times \\ &\times\sum_{i=2}^{an-3}\sum_{j=4}^{an-1}\left\{\left(1-\frac{4(1-p)}{n-1}\right)^{i}\left(1-\frac{3(1-p)}{n-1}\right)^{j-i}\left(1-\frac{2(1-p)}{n-1}\right)^{-j}\right\}\end{aligned}$$

Simplifying, we get

$$P(L_2) = \left(1 - \frac{4}{n}\right)\left(\frac{1-p}{n-1}\right)^2 \left(1 - \frac{3(1-p)}{n-1}\right)^{-2} \left(1 - \frac{4(1-p)}{n-1}\right)^{-2} \times \\ \times \left(1 - p - \frac{1-p}{n-1}\right)\left(1 - p - \frac{2(1-p)}{n-1}\right)\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1} z_2,$$

where

$$\begin{aligned} z_2 &= \sum_{i=2}^{an-3} \sum_{j=i+2}^{an-1} \left\{ \left(1 - \frac{4(1-p)}{n-1}\right)^i \left(1 - \frac{3(1-p)}{n-1}\right)^{-i} \left(1 - \frac{3(1-p)}{n-1}\right)^j \left(1 - \frac{2(1-p)}{n-1}\right)^{-j} \right\} \\ &= \sum_{i=2}^{an-3} \sum_{j=i+2}^{an-1} \left\{ \left(\frac{n+4p-5}{n+3p-4}\right)^i \left(\frac{n+3p-4}{n+2p-3}\right)^j \right\} \\ &= \sum_{i=2}^{an-3} \sum_{j=i+2}^{an-1} \left\{ \left(1 - \frac{1-p}{n+3p-4}\right)^i \left(1 - \frac{1-p}{n+2p-3}\right)^j \right\}. \end{aligned}$$

Note that, the only sequence which contributes here has the freedom of placing the occurring nodes *almost* anywhere in the input sequence while satisfying the requiring conditions. This follows in all subsequent cases.

(4) L_3: The input sequence for L_3 has the following conditions, no v_1 till EoS, one v_n till EoS, one v_{n-1} till v_n arrives, one v_{n-2} till v_{n-1} arrives, and no v_{n-3} till v_{n-2} arrives.

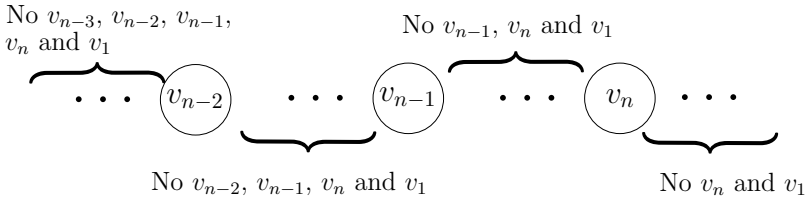


Figure 5.6: A visualization of input sequence for L_3.

Similarly, for leaf L_3 the contributing input sequence has v_{n-2}, v_{n-1} and v_n at 3 independent positions, i, j and k. By Lemmas 25 and 26, the limiting value of the probability of the rest of the sequences is 0. For the former sequence with 3 independent positions, the probability of the

input sequence for fixed i, j and k is:

$$\begin{aligned}
\underset{i,j,k \text{ fixed}}{P(L_3)} &= \left(1-\frac{5}{n}\right)\left(1-\frac{5(1-p)}{n-1}\right)^{i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{3(1-p)}{n-1}\right)\times \\
&\times\left(1-\frac{4(1-p)}{n-1}\right)^{j-i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{2(1-p)}{n-1}\right)\times \\
&\times\left(1-\frac{3(1-p)}{n-1}\right)^{k-j-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{1-p}{n-1}\right)\times \\
&\times\left(1-\frac{2(1-p)}{n-1}\right)^{an-k-1}
\end{aligned}$$

And the probability of L_3 for all possible positions of i, j and k is

$$\begin{aligned}
P(L_3) = \sum_{i=2}^{an-5}\sum_{j=i+2}^{an-3}\sum_{k=j+2}^{an-1}\Bigg\{&\left(1-\frac{5}{n}\right)\times \\
&\times\left(1-\frac{5(1-p)}{n-1}\right)^{i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{3(1-p)}{n-1}\right)\times \\
&\times\left(1-\frac{4(1-p)}{n-1}\right)^{j-i-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{2(1-p)}{n-1}\right)\times \\
&\times\left(1-\frac{3(1-p)}{n-1}\right)^{k-j-2}\left(\frac{1-p}{n-1}\right)\left(1-p-\frac{1-p}{n-1}\right)\times \\
&\times\left(1-\frac{2(1-p)}{n-1}\right)^{an-k-1}\Bigg\}
\end{aligned}$$

This can be further simplified as

$$\begin{aligned}
P(L_3) = &\left(1-\frac{5}{n}\right)\left(\frac{1-p}{n-1}\right)^3\left(1-\frac{3(1-p)}{n-1}\right)^{-2}\left(1-\frac{4(1-p)}{n-1}\right)^{-2}\left(1-\frac{5(1-p)}{n-1}\right)^{-2}\times \\
&\times\left(1-p-\frac{1-p}{n-1}\right)\left(1-p-\frac{2(1-p)}{n-1}\right)\left(1-p-\frac{3(1-p)}{n-1}\right)\left(1-\frac{2(1-p)}{n-1}\right)^{an-1} z_3,
\end{aligned}$$

where

$$\begin{aligned}
z_3 = &\sum_{i=2}^{an-5}\sum_{j=i+2}^{an-3}\sum_{k=j+2}^{an-1}\left\{\left(1-\frac{5(1-p)}{n-1}\right)^i\left(1-\frac{4(1-p)}{n-1}\right)^{j-i}\left(1-\frac{3(1-p)}{n-1}\right)^{k-j}\right\} \\
&\sum_{i=2}^{an-5}\sum_{j=i+2}^{an-3}\sum_{k=j+2}^{an-1}\left\{\left(1-\frac{1-p}{n+4p-5}\right)^i\left(1-\frac{1-p}{n+3p-4}\right)^j\left(1-\frac{1-p}{n+2p-3}\right)^k\right\}.
\end{aligned}$$

Similarly, we can write the probability of a leaf node at an arbitrary k^{th} level, L_k, where k varies from 0 to $n-1$.

$$P(L_k) = \left(1 - \frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k \prod_{i=1}^{k}\left\{\left(1 - p - \frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1 - \frac{j(1-p)}{n-1}\right)^{-2}\right\}\times$$
$$\times\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1} z_k,$$

where z_k is

$$z_k = \sum_{i_1=2}^{an-2k+1}\sum_{i_2=i_1+2}^{an-2k+3}\cdots\sum_{\substack{i_{k-1}=\\ i_{k-2}+2}}^{an-3}\sum_{\substack{i_k=\\ i_{k-1}+2}}^{an-1}\left\{\left(1 - \frac{1-p}{n+(k+1)p-(k+2)}\right)^{i_1}\times\right.$$
$$\left.\times\left(1 - \frac{1-p}{n+kp-(k+1)}\right)^{i_2}\cdots\left(1 - \frac{1-p}{n+3p-4}\right)^{i_{k-1}}\left(1 - \frac{1-p}{n+2p-3}\right)^{i_k}\right\}$$

Simplifying $P(L_k)$

The probability of the event at k^{th} level, given by $P(L_k)$ is a complex term, involving nested summations and different terms with varying exponent. We can simplify it a little further and introduce similar expressions to upper and lower bound it. We observe that the limiting value of the upper and lower bounds calculated in the previous step coincides, which in turn gives us the limiting value of $P(L_k)$.

$P(L_k)$ is equal to

$$P(L_k) = \left(1 - \frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k \prod_{i=1}^{k}\left\{\left(1 - p - \frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1 - \frac{j(1-p)}{n-1}\right)^{-2}\right\}\times \tag{27}$$
$$\times\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1} z_k, \tag{28}$$

where z_k is

$$z_k = \sum_{i_1=2}^{an-2k+1} \sum_{i_2=i_1+2}^{an-2k+3} \cdots \sum_{\substack{i_{k-1}= \\ i_{k-2}+2}}^{an-3} \sum_{\substack{i_k= \\ i_{k-1}+2}}^{an-1} \left\{ \left(1 - \frac{1-p}{n+(k+1)p-(k+2)}\right)^{i_1} \times \right.$$
$$\left. \times \left(1 - \frac{1-p}{n+kp-(k+1)}\right)^{i_2} \cdots \left(1 - \frac{1-p}{n+3p-4}\right)^{i_{k-1}} \left(1 - \frac{1-p}{n+2p-3}\right)^{i_k} \right\}$$

where for now, we focus on the last part of the equation, z_k.

z_k is a term containing k consecutive nested summations. The upper bounds of the summation have terms like $-2k+1, -2k+3 \cdots -1$ constant with respect to summation index i, and can be taken out of the summation bounds by introducing extra summands. Shifting the lower and upper bounds of summation, we obtain

$$z_k = \sum_{i_1=0}^{an-2k+1} \sum_{i_2=i_1}^{an-2k+1} \cdots \sum_{i_{k-1}=i_{k-2}}^{an-2k+1} \sum_{i_k=i_{k-1}}^{an-2k+1} \left\{ \left(1 - \frac{1-p}{n+(k+1)p-(k+2)}\right)^{i_1+2k} \times \right.$$
$$\left. \times \left(1 - \frac{1-p}{n+kp-(k+1)}\right)^{i_2+2(k-1)} \cdots \left(1 - \frac{1-p}{n+3p-4}\right)^{i_{k-1}+4} \left(1 - \frac{1-p}{n+2p-3}\right)^{i_k+2} \right\}$$

The fractions in the nested summation hold the following inequality

$$\left(1 - \frac{1-p}{n+(k+1)p-(k+2)}\right) \leq \left(1 - \frac{1-p}{n+2p-3}\right)$$

We can introduce an upper bound and lower bound for z_k, by replacing all intermediate summands by either of these two fractions.

Using the following notations for the fractions,

$$f_u(n,p,k) = f_u = \left(1 - \frac{1-p}{n+2p-3}\right)$$

and

$$f_l(n,p,k) = f_l = \left(1 - \frac{1-p}{n+(k+1)p-(k+2)}\right)$$

We thus obtain an upper bound uz_k as

$$uz_k = \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} \left\{ f_u^{i_1+2k} \cdot f_u^{i_2+2(k-1)} \cdots f_u^{i_{k-1}+4} \cdot f_u^{i_k+2} \right\}$$

and a lower bound lz_k as

$$lz_k = \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} \left\{ f_l^{i_1+2k} \cdot f_l^{i_2+2(k-1)} \cdots f_l^{i_{k-1}+4} \cdot f_l^{i_k+2} \right\}$$

By taking out the constants terms in the nested summation, specifically the power of summand $f_u^{2k}, f_u^{2(k-1)}, \ldots$ and $f_l^{2k}, f_l^{2(k-1)}, \ldots$ the expressions can be further simplifies as

$$z_k \leq uz_k = \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} f_u^{\sum_{j=1}^{j=k} i_j + 2\cdot\frac{k(k+1)}{2}} = f_u^{k(k+1)} \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} f_u^{\sum_{j=1}^{j=k} i_j} \tag{29}$$

and

$$lz_k = \sum_{i_1=0}^{an-2k+1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} f_l^{\sum_{j=1}^{j=k} i_j + 2\cdot\frac{k(k+1)}{2}} = f_l^{k(k+1)} \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} f_l^{\sum_{j=1}^{j=k} i_j} \leq z_k \tag{30}$$

We have obtained two nested summations, the upper and lower bounds, where the interior summand terms are the same. We can now introduce a lemma that specifies replacing this nested summation with a *Gaussian binomial coefficient*.

Lemma 27. *The coefficient of a function f under nested summations can be expressed in terms of* partition function $p(N, k, s)$*, where $p(N, k, s)$ counts the total number of ways of partitioning a number s with at most k number of parts, and under the constraints that the maximum size of any partition is N.*

$$\underset{(k \text{ nested sums})}{\sum_{i_1=0}^{N} \cdots \sum_{i_k=I_{k-1}}^{N}} f^{i_1+i_2+\ldots i_k} = \sum_{s=0}^{N\cdot k} p(N, k, s).f^s$$

$$\sum_{s=0}^{N\cdot k} p(N, k, s).f^s = \binom{N+k}{k}_f,$$

where $\binom{N+k}{k}_f$ *is the Gaussian Binomial Coefficient, and is equal to*

$$\binom{N+k}{k}_f = \frac{(1-f^{N+k})(1-f^{N+k-1})\cdots(1-f^{N+1})}{(1-f)(1-f^2)\cdots(1-f^k)}\ .$$

We use this lemma to simplify the upper bound uz_k and lower bound lz_k and we obtain

$$\begin{aligned} uz_k &= f_u^{k(k+1)} \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} f_u^{\sum_{j=1}^{j=k} i_j} = f_u^{k(k+1)} \sum_{i=0}^{(an-2k-1)\cdot k} p(an-2k-1,k,s).(f_u)^s \\ &= f_u^{k(k+1)} \binom{(an-2k-1)+k}{k}_{f_u} = f_u^{k(k+1)} \binom{an-k-1}{k}_{f_u} \end{aligned}$$

and

$$\begin{aligned} lz_k &= f_l^{k(k+1)} \sum_{i_1=0}^{an-2k-1} \cdots \sum_{i_k=i_{k-1}}^{an-2k-1} f_l^{\sum_{j=1}^{j=k} i_j} = f_l^{k(k+1)} \sum_{i=0}^{(an-2k-1)\cdot k} p(an-2k-1,k,s).(f_l)^s \\ &= f_l^{k(k+1)} \binom{(an-2k-1)+k}{k}_{f_l} = f_l^{k(k+1)} \binom{an-k-1}{k}_{f_l} \end{aligned}$$

Calculating the limits for the upper and lower bound

In this step, we make use of inequalities individually on uz_k and lz_k to derive simpler terms which are greater and lesser than uz_k and lz_k respectively. We substitute these new, simpler terms for z_k in the R.H.S. of $P(L_k)$ and hence we get upper and lower bounds on the value of $P(L_k)$ respectively. These bounds coincide, and hence we get the limiting value of the $P(L_k)$ term.

Inequality for Upper Bound uz_k:

First, we evaluate the upper bound as follows.

$$uz_k = (f_u)^{k(k+1)} \binom{an-k-1}{k}_{f_u} = (f_u)^{k(k+1)} \cdot \frac{(1-f_u^{an-k-1})(1-f_u^{an-k-2})\cdots(1-f_u^{an-2k})}{(1-f_u)(1-f_u^2)\cdots(1-f_u^k)} \tag{31}$$

We introduce a lemma to further upper bound this expression.

Lemma 28.

$$\frac{1}{(1-f_u)(1-f_u^2)\cdots(1-f_u^k)} \leq \frac{n^k}{k!\cdot(1-p)^k}\cdot h_n(k,p) \tag{32}$$

Proof. The following inequality holds from [56].

$$\begin{aligned}(1+x)^k &\leq \frac{1}{1-kx} \quad \text{for } x\in[-1,1/k),\ k\geq 0\\ 1-(1+x)^k &\geq 1-\frac{1}{1-kx}\\ \frac{1}{1-(1+x)^k} &\leq \frac{1}{1-\frac{1}{1-kx}} = 1-\frac{1}{kx}\end{aligned}$$

For $x=-\frac{1-p}{n+2p-3}$, $-1\leq x\leq 1/k$, and $kx=\frac{-k(1-p)}{n+2p-3}$,

$$1-\frac{1}{kx} = 1+\frac{n+2p-3}{k(1-p)} = \frac{k(1-p)+n+2p-3}{k(1-p)}$$

Using this to simplify the denominator of the fraction in Equation 31, we get

$$\begin{aligned}&\frac{1}{(1-f_u)(1-f_u^2)\cdots(1-f_u^k)} \leq\\ &\leq \left(\frac{k(1-p)+n+2p-3}{k(1-p)}\right)\left(\frac{(k-1)(1-p)+n+2p-3}{(k-1)(1-p)}\right)\cdots\left(\frac{1(1-p)+n+2p-3}{1(1-p)}\right)\\ &\leq \frac{n^k}{k!\cdot(1-p)^k}\cdot\left(\frac{k(1-p)+n+2p-3}{n}\right)\left(\frac{(k-1)(1-p)+n+2p-3}{n}\right)\cdots\left(\frac{1(1-p)+n+2p-3}{n}\right)\\ &\leq \frac{n^k}{k!\cdot(1-p)^k}\cdot h_n(k,p)\end{aligned} \tag{33}$$

□

Inequality for Lower Bound lz_k:

Similarly, we evaluate the lower bound as follows.

$$lz_k = (f_l)^{k(k+1)}\binom{an-k-1}{k}_{f_l} = (f_l)^{k(k+1)}\cdot\frac{(1-f_l^{an-k-1})(1-f_l^{an-k-2})\cdots(1-f_l^{an-2k})}{(1-f_l)(1-f_l^2)\cdots(1-f_l^k)} \tag{34}$$

We introduce the following lemma to lower bound this expression.

Lemma 29.

$$\frac{1}{(1-f_l)(1-f_l^2)\cdots(1-f_l^k)} \geq \frac{n^k}{k!\cdot(1-p)^k}\cdot g_n(k,p) \tag{35}$$

Proof. The following inequality holds from [56].

$$1+kx \leq (1+x)^k \quad \text{for } x \geq -1,\ k \geq 0$$

$$1-(1+kx) \geq 1-(1+x)^k$$

$$\frac{1}{1-(1+kx)} = \frac{-1}{kx} \leq \frac{1}{1-(1+x)^k}$$

For $x = -\frac{1-p}{n+(k+1)p-(k+2)}$, $x \geq -1$, and $kx = \frac{-k(1-p)}{n+2p-3}$

$$\frac{-1}{kx} = \frac{n+(k+1)p-(k+2)}{k(1-p)}$$

Using this to simplify the denominator of the fraction in Equation 34, we get

$$\begin{aligned}
&\frac{1}{(1-f_l)(1-f_l^2)\cdots(1-f_l^k)} \geq \\
&\geq \left(\frac{n+(k+1)p-(k+2)}{k(1-p)}\right)\left(\frac{n+kp-(k+1)}{(k-1)(1-p)}\right)\cdots\left(\frac{n+2p-3}{1(1-p)}\right) \geq \\
&\geq \frac{n^k}{k!\cdot(1-p)^k}\cdot\left(\frac{n+(k+1)p-(k+2)}{n}\right)\left(\frac{n+kp-(k+1)}{n}\right)\cdots\left(\frac{n+2p-3}{n}\right) \\
&\geq \frac{n^k}{k!\cdot(1-p)^k}\cdot g_n(k,p)
\end{aligned} \tag{36}$$

□

We have obtained two expressions – one that is greater than the upper bound of z_k and one that is smaller than the lower bound of z_k. We can now substitute the value of z_k with these terms in $P(L_k)$ and calculate the upper and lower bound of $P(L_k)$.

Substituting the results of Lemma 28 in the upper bound of $P(L_k)$ from Equation 29 gives us:

(For upper bound of z_k)

$$\underset{\text{upper bound}}{P(L_k)} = \left(1 - \frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k \prod_{i=1}^{k}\left\{\left(1 - p - \frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1 - \frac{j(1-p)}{n-1}\right)^{-2}\right\}\times$$
$$\times\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1}\cdot(f_u)^{k(k+1)}\cdot\frac{(1-f_u^{an-k-1})(1-f_u^{an-k-2})\cdots(1-f_u^{an-2k})}{(1-f_u)(1-f_u^2)\cdots(1-f_u^k)}.$$

$$\underset{\text{upper bound}}{P(L_k)} \leq \left(1 - \frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k \prod_{i=1}^{k}\left\{\left(1 - p - \frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1 - \frac{j(1-p)}{n-1}\right)^{-2}\right\}\times$$
$$\times\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1}\left(1 - \frac{1-p}{n+2p-3}\right)^{k(k+1)}\left(1 - \left(1 - \frac{1-p}{n+2p-3}\right)^{an-k-1}\right)\times$$
$$\times\left(1 - \left(1 - \frac{1-p}{n+2p-3}\right)^{an-k-2}\right)\cdots\left(1 - \left(1 - \frac{1-p}{n+2p-3}\right)^{an-2k}\right)\frac{n^k}{k!\cdot(1-p)^k}\cdot h_n(k,p)$$

We calculate the individual limits of the terms in the R.H.S. and use product law to calculate the limit of the term in the L.H.S.

(1)

$$\lim_{n\to\infty}\left(1 - \frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k\frac{n^k}{k!\cdot(1-p)^k}\cdot h_n(k,p) = \frac{1}{k!}$$

The limit of $h_n(k,p)$ as n goes to ∞ is 1.

(2)

$$\lim_{n\to\infty}\prod_{i=1}^{k}\left\{\left(1 - p - \frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1 - \frac{j(1-p)}{n-1}\right)^{-2}\right\} = (1-p)^k$$

(3)

$$\left(1 - \frac{2(1-p)}{n-1}\right)^{an-1}\left(1 - \frac{1-p}{n+2p-3}\right)^{k(k+1)} = e^{-2a(1-p)}$$

(4)

$$\lim_{n\to\infty}\left(1 - \left(1 - \frac{1-p}{n+2p-3}\right)^{an-k-1}\right)\left(1 - \left(1 - \frac{1-p}{n+2p-3}\right)^{an-k-2}\right)\times$$
$$\times\cdots\left(1 - \left(1 - \frac{1-p}{n+2p-3}\right)^{an-2k}\right) = \left(1 - e^{-a(1-p)}\right)^k$$

Combining these terms, we get the limit to be

$$\lim_{n\to\infty} \underset{\text{upper bound}}{P(L_k)} \leq \frac{\left(\left(1-e^{-a(1-p)}\right)\cdot(1-p)\right)^k}{k!}\cdot e^{-2a(1-p)}$$

Substituting the results of Lemma 29 in the lower bound of $P(L_k)$ from Equation 30 gives us:

(For lower bound of z_k)

$$\underset{\text{lower bound}}{P(L_k)} = \left(1-\frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k \prod_{i=1}^{k}\left\{\left(1-p-\frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1-\frac{j(1-p)}{n-1}\right)^{-2}\right\}\times$$
$$\times\left(1-\frac{2(1-p)}{n-1}\right)^{an-1}\cdot(f_l)^{k(k+1)}\cdot\frac{(1-f_l^{an-k-1})(1-f_l^{an-k-2})\cdots(1-f_l^{an-2k})}{(1-f_l)(1-f_l^2)\cdots(1-f_l^k)}.$$

$$\underset{\text{lower bound}}{P(L_k)} \geq \left(1-\frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k \prod_{i=1}^{k}\left\{\left(1-p-\frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1-\frac{j(1-p)}{n-1}\right)^{-2}\right\}\times$$
$$\times\left(1-\frac{2(1-p)}{n-1}\right)^{an-1}\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{k(k+1)}\times$$
$$\times\left(1-\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{an-k-1}\right)\left(1-\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{an-k-2}\right)\times$$
$$\times\cdots\left(1-\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{an-2k}\right)\frac{n^k}{k!\cdot(1-p)^k}\cdot g_n(k,p)$$

Once again, we use product law of limits here to calculate the limiting value of the L.H.S of the expression by calculating individual limits on the terms in R.H.S.

(1)

$$\lim_{n\to\infty}\left(1-\frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^k\frac{n^k}{k!\cdot(1-p)^k}\cdot g_n(k,p) = \frac{1}{k!}$$

The limit of $g_n(k,p)$ as n goes to ∞ is 1.

(2)

$$\lim_{n\to\infty}\prod_{i=1}^{k}\left\{\left(1-p-\frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1-\frac{j(1-p)}{n-1}\right)^{-2}\right\} = (1-p)^k$$

(3)

$$\lim_{n\to\infty}\left(1-\frac{2(1-p)}{n-1}\right)^{an-1}\cdot\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{k(k+1)}=e^{-2a(1-p)}$$

(4)

$$\lim_{n\to\infty}\left(1-\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{an-k-1}\right)\left(1-\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{an-k-2}\right)\times$$
$$\times\cdots\left(1-\left(1-\frac{1-p}{n+(k+1)p-(k+2)}\right)^{an-2k}\right)=\left(1-e^{-a(1-p)}\right)^{k}$$

Combining these terms, we get the limit to be

$$\lim_{n\to\infty}\underset{\text{lower bound}}{P(L_k)}\geq\frac{\left(\left(1-e^{-a(1-p)}\right)\cdot(1-p)\right)^{k}}{k!}\cdot e^{-2a(1-p)}$$

As both these bounds coincide, using the squeeze or the sandwich theorem we get that

$$\lim_{n\to\infty}P(L_k)=\frac{\left(\left(1-e^{-a(1-p)}\right)\cdot(1-p)\right)^{k}}{k!}\cdot e^{-2a(1-p)}$$

Step 2: Defining an upper bound on $P(L_k)$ and proving the infinite summation converges

Till here, we have found the limit of $P(L_k)$. Now, we find to find an M_k such that $P(L_k)\leq M_k$, and $\sum_{k=0}^{\infty}M_k\leq\infty$.

We define an M_k next using the expression in Equation 27, when we first expressed $P(L_k)$.

$$P(L_k)=\left(1-\frac{k+2}{n}\right)\left(\frac{1-p}{n-1}\right)^{k}\prod_{i=1}^{k}\left\{\left(1-p-\frac{k(1-p)}{n-1}\right)\right\}\prod_{j=3}^{k+2}\left\{\left(1-\frac{j(1-p)}{n-1}\right)^{-2}\right\}\times$$
$$\times\left(1-\frac{2(1-p)}{n-1}\right)^{an-1}z_k,$$

where z_k is

$$z_k = \sum_{i_1=2}^{an-2k+1} \sum_{i_2=i_1+2}^{an-2k+3} \cdots \sum_{\substack{i_{k-1}= \\ i_{k-2}+2}}^{an-3} \sum_{\substack{i_k= \\ i_{k-1}+2}}^{an-1} \left\{ \left(1 - \frac{1-p}{n+(k+1)p-(k+2)}\right)^{i_1} \times \right.$$
$$\left. \times \left(1 - \frac{1-p}{n+kp-(k+1)}\right)^{i_2} \cdots \left(1 - \frac{1-p}{n+3p-4}\right)^{i_{k-1}} \left(1 - \frac{1-p}{n+2p-3}\right)^{i_k} \right\}$$

We can upper bound the value of $P(L_k)$ here using these inequalities. Note, all the internal terms with the nested summation in z_k are less than 1, so the expression reduces to

$$z_k \leq \sum_{i_1=2}^{an-2k+1} \sum_{i_2=i_1+2}^{an-2k+3} \cdots \sum_{\substack{i_{k-1}= \\ i_{k-2}+2}}^{an-3} \sum_{\substack{i_k= \\ i_{k-1}+2}}^{an-1} 1^{i_1+i_2+\ldots i_k}$$

which is equal to $\binom{an-2k-1+k}{k}_1$. In case where $f = 1$ for Gaussian binomial coefficient $\binom{N+k}{k}_f$, it becomes equal to the normal binomial coefficient, $\binom{N+k}{k}$. Using the same logic here, we can state $z_k = \binom{an-2k-1+k}{k} = \binom{an-k-1}{k} \leq \frac{n^k}{k!}$.

The two product terms in $P(L_k)$ are upper bounded by 1. Hence, $P(L_k)$ can be upper bounded as follows:

$$\begin{aligned} P(L_k) &\leq 1 \cdot \left(\frac{1-p}{n-1}\right)^k \cdot 1 \cdot e^{-2a(1-p)} \cdot \frac{n^k}{k!} \\ &\leq \left(\frac{n}{n-1}\right)^k \cdot e^{-2a(1-p)} \cdot \frac{(1-p)^k}{k!} \\ &\leq e^k \cdot e^{-2a(1-p)} \cdot \frac{(1-p)^k}{k!} \end{aligned}$$

This term, $e^k \cdot e^{-2a(1-p)} \cdot \frac{(1-p)^k}{k!}$ is the required M_k and the infinite sum of this term,

$$\sum_{k=0}^{\infty} e^{-2a(1-p)} \cdot \frac{e^k (1-p)^k}{k!} = e^{e(1-p)}$$

which converges, where the summation follows from the expansion of Taylor series, $\sum_{k=0}^{\infty} \frac{x^k}{k!} = e^k$. This satisfies the required conditions for Tannery's theorem.

Step 3: Exchanging the order of limits and summation of $P(L_k)$

We have shown the conditions for interchanging the order of sum and limits are satisfied with respect to Tannery's theorem. We now calculate the infinite sum of limits of the limiting value of $P(L_k)$, and we get

$$\begin{aligned}
\lim_{n\to\infty}\sum_{k=0}^{n-1} P(L_k) &= \sum_{k=0}^{\infty}\lim_{n\to\infty} P(L_k) \\
&= \sum_{k=0}^{\infty} \frac{\left(\left(1-e^{-a(1-p)}\right)\cdot(1-p)\right)^k}{k!}\cdot e^{-2a(1-p)} \\
&= e^{-2a(1-p)}\cdot\sum_{k=0}^{\infty}\frac{\left(\left(1-e^{-a(1-p)}\right)\cdot(1-p)\right)^k}{k!} \\
&= e^{-2a(1-p)}\cdot e^{(1-p)\cdot\left(1-e^{-a(1-p)}\right)} \\
&= e^{(1-p)(1-2a-e^{-a(1-p)})}
\end{aligned}$$

where the summation follows from expansion of Taylor series, $\sum_{k=0}^{\infty}\frac{x^k}{k!} = e^k$.

Therefore we obtain the result,

$$\lim_{n\to\infty}\mathbb{E}[X_1] = \lim_{n\to\infty}\sum_{k=0}^{n-1} P(L_k) = e^{(1-p)(1-2a-e^{-a(1-p)})}$$

5.3.5 Tight Bounds on Competitive Ratio

By computing the probability of desired sequences above, now we can calculate the size of expected matching for the case of $TSM-A$. We express the result in Theorem 30 as follows.

Theorem 30.

$$\lim_{n\to\infty}\frac{\mathbb{E}[X_1]}{n} = 1 - e^{(1-p)(1-2a-e^{-a(1-p)})} \tag{37}$$

Using Theorems 13 and 30, we conclude that the asymptotic competitive ratio of Two Suggested Matching - Non Adaptive algorithm is:

$$\rho_{TSM-A} = \frac{1-e^{(1-p)(1-2a-e^{-a(1-p)})}}{1-\frac{e^{2(ap-a)}}{1-(e^{ap-a}\cdot a(1-p)^2)}} \tag{38}$$

Once again, we present the results for some specific parameters of a and p.

Corollary 31. *For the conventional lazy random walk based Markov chain, where $p = \frac{1}{2}$, we obtain the expected size of matching as:*

$$\rho_{TSM-A} = \frac{1 - e^{\frac{1}{2}\cdot\left(1-2a-e^{\frac{-a}{2}}\right)}}{1 - \frac{4}{4e^a - ae^{\frac{a}{2}}}}$$

Corollary 32. *For the conventional lazy random walk Markov chain, where $p = \frac{1}{2}$, when the input sequence length is equal to the size of (a single) vertex set or $a = 1$, we get the expected size of matching as:*

$$\rho_{TSM-A} = \frac{1 - e^{-\frac{(1+\sqrt{e})}{2\sqrt{e}}}}{1 - \frac{4}{4e-\sqrt{e}}} = \frac{1 - 0.4478}{0.56636} = \frac{0.5521}{0.56636}$$

$$\rho_{TSM-A} = 0.9748$$

Hence, under the lazy random walk Markov chain, $p = \frac{1}{2}$, Two Suggested Matching - Adaptive achieves a competitive ratio of $\rho_{TSM-A} = 0.9748$.

5.4 Summary of Results

We conclude the chapter by summarizing the size of expected matchings and competitive ratios obtained above for the case of the conventional lazy random walk based Markov chain, where $p = \frac{1}{2}$. For this specific Markov chain, we obtained the following competitive ratios:

$$\rho_{TSM-NA} = \frac{(4 - 4e^{-a} - ae^{-a})(4e^a - ae^{\frac{a}{2}})}{4 \cdot \left((4e^a - ae^{\frac{a}{2}}) - 4\right)}$$

and,

$$\rho_{TSM-A} = \frac{\left(1 - e^{\frac{1}{2}\cdot\left(1-2a-e^{\frac{-a}{2}}\right)}\right)(4e^a - ae^{\frac{a}{2}})}{(4e^a - ae^{\frac{a}{2}}) - 4}$$

We wanted to analyze how the competitive ratio behaves as a function of a, where a is the factor by which we scale the size of input, with respect to the graph size n. We plotted the values of the

competitive ratio of both these algorithms by increasing the value of a from 0.1 to 10.6, with an addendum of 0.1 at each step, and Fig 5.7 shows the change in competitive ratio for such.

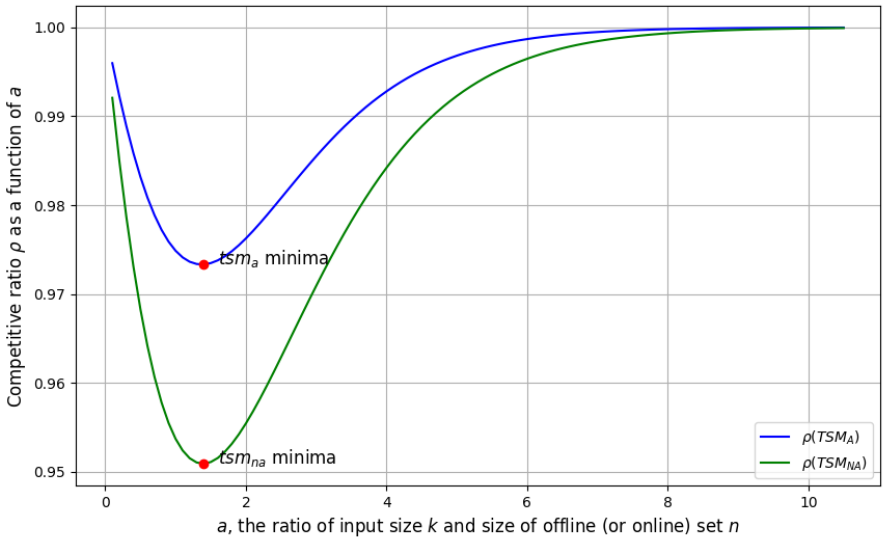


Figure 5.7: Graph plotting the changes in competitive ratio of $TSM-NA$ and $TSM-A$ as a function of a.

Note that the competitive ratios approach their minimum value for both algorithm near the range of $a = 1.35$ to 1.4. From further investigations, we derived the following corollary.

Corollary 33. *$TSM-NA$ achieves a competitive ratio of at least* 0.9509 *in online bipartite matching in* $(2,2)$*-regular bipartite graph under the lazy random walk Markov chain model, where* $p = \frac{1}{2}$. *$TSM-A$ achieves a competitive ratio of at least* 0.9733 *in online bipartite matching in* $(2,2)$*-regular bipartite graph under the lazy random walk Markov chain model, where* $p = \frac{1}{2}$.

Proof. To analyze further where the competitive ratio becomes minimized for the respective values of a in both of these algorithms, we solve by setting the derivative of the competitive ratio with respect to a to be 0. We get the following results:

For $TSM-NA$, $\frac{d\ (\rho_{TSM-NA})}{da} = 0$, we get $a \approx 1.3896$. And for the case of $TSM-A$, setting $\frac{d\ (\rho_{TSM-A})}{da}$, where $f(a) = \rho_{TSM-A}$, we obtain $a \approx 1.3714$. By plugging in these values, we see that for $p = \frac{1}{2}$-lazy random walk Markov chain setting of online bipartite matching, $TSM-NA$ achieves the worst competitive ratio of $\rho_{TSM-NA} = 0.9509$ at $a \approx 1.3896$, and $TSM-A$ achieves the worst

competitive ratio of $\rho_{TSM-A} = 0.9733$ at $a \approx 1.3714$. □

Next, we delve into experiments based on the statistics of a real world online ad allocation matching competition which helped in mapping out the underlying type graph and Markov chain on which our results and analysis are based.

Chapter 6

Experimental Analysis of Real World Online Bipartite Matching Data

As stated previously, this thesis is the first work which considers dependency among online vertices in the problem of online bipartite matching. The problem for general type graphs and Markov chains presented challenges in terms of complexity and unclear relationship between their parameters. To gain a clearer understanding and investigate the issue, we turned to real world data where this dependency is observed and can be replicated. This chapter briefly summarizes our findings and presents the experimental setup that led to the development of type graphs and Markov chains specific to our thesis.

6.1 Introduction

The experiment discussed in the chapter serves as the basis of the type graph and Markov chains components of the matching problem we discuss in this thesis. We looked through different experiments and data-sets which can be abstracted to model the online bipartite matching problem. The intention of looking into real world data was to better understand the natural dependency which occurs among online vertices. We prefixed our search to find experiments where - (1) The procedure revolved around the pairing of two opposite entities, for example, real time advertisements, auction bidding, cab request service. (2) We would be able to extract the underlying bipartite graph

structure from the experiment, and largely throughout the experiment, the graph structure remains non evolving. A few experiments had a highly evolving and time-varying graph structure, e.g. cab service requests. (3) Lastly, we wanted experiments where the input data occurred in an online fashion, and we focused on experiments which had either discrete time steps capturing the evolving system, or data incorporating time stamps. This was needed to learn how the current data set was dependent on the previous one, and hence, can be used to construct a Markov chain, using simpler approaches like frequency analysis.

Keeping these in mind, we begin our research by looking into real world examples of online matching where the input has a stochastic taste to it. We surveyed various data-sets, including but not limited to auction bidding [76], cab service requests [78], patient oxygen demands during COVID-19 [77], and online advertisement bidding [75], [74]. We tried to obtain the data sets behind doctor residency matching, in which incoming doctors are matched or allotted to residency programs. Despite being one of the most famous and excellent use cases of bipartite matching besides online ad allocation, data sets related to this were extremely tough to obtain, possibly due to privacy regulations and data protection laws. The most appropriate data set and experiment, which suited the characterization of Markov chain was the iPinYou Real Time Bidding Data set [75]. In the following sections, we briefly explain the original experiment, and how we extracted the type graph and Markov chain from the data set.

6.2 iPinYou Real-Time Bidding Dataset

iPinYou is a prominent Chinese marketing technology company, which among other things, provides digital advertising solutions. Of particular interest is the *Demand Side Platform* (DSP) of iPinYou. A DSP is a service or software which provides automation for buying and selling ad impressions in real time. In other words, DSPs let advertisers manage their target audience, budget, ad formats, and bidding amounts for showcasing an ad to an impression, along with providing data metrics and analytical tools to optimize their campaign. A DPS works in conjugation with a *Real-Time Bidding* (RTB) algorithm employed by the advertiser end.

An RTB algorithm provides a mechanism to automate the process of bidding for an impression

on behalf of the advertisers. Impressions arrive in online fashion, when let's say a user searches a query, or visits a popular website that showcases advertisements. The impression carries data like browser and network specific information (browser model, web engine model, IP address, etc.), user specific information (e.g. gender, age, likes-dislikes), date and time, query, site visited, etc. This helps in deciding the advertisers or the RTB algorithm to decide if it should bid to display ad to this impression, and what percentage of budget to allocate for this bid, etc. Once the auction begins, different RTBs representing different advertisers strategically submit their bids for each impression to the bidding component of the DSP, and DSP decides the winner of the bid, and notifies the top bidder. The winning RTB then displays the advertisement to the user corresponding to the impression. So much happens, in such a small time when a user just submits a query or visits a website! For this to efficiently take place, designing fast and efficient RTB algorithms, as well as DSPs are important.

In 2013, iPinYou organized a global Real-Time Bidding algorithm competition spanning three seasons, where each season included both an offline and an online stage. During the offline stage, iPinYou provided a dataset for model training, while reserving another dataset for testing purposes. The datasets contained logs of ad bids, impressions, clicks, and conversions. Upon the competition's conclusion, iPinYou made these datasets publicly available. This marked the release of the first publicly accessible dataset related to RTB display advertising. The datasets serve as valuable resources for addressing key research challenges, such as bid optimization and Click-Through Rate (CTR) estimation.

We concentrate on the bidding logs of the advertisers in the experimental setup. Using the offline stage bidding logs data set, we gather which advertisers are interested in what types of impressions. We approximate an impression using the same information which is supplied to the RTB, browser information, user information, and so forth.

Two data elements, 'iPinYouId' and 'adSlotId' were sufficient in comprehensively identifying the impression. These impressions were bid upon by 5 advertisers, classifying the impressions into groups of $2^5 - 1$ sets, with each set denoting a subset of advertisers interested in them. Using these, we were able to extract the underlying bipartite type graph between the 5 advertisers (which formed the offline vertex set) and the 31 impression types. An edge was present between them if the

advertisers had bid for that particular type of impression.

Along with the browser, site, and user details in the logs, another data slot given was the date and time for when the bid was supplied for the impression. This was a critical aspect in tracking the arrival of impressions. We chronologically sorted the impressions and used frequency analysis to extract a Markov chain on these impression types.

We noticed that each bidding log contained the bids for the whole duration of the day or 24 hours period, and larger log files had about 4.7 million entries. Such log files led to the composition of very general Markov chains, in which identifying a structure or some theoretic properties was very tough. Moreover, a period of 24 hours is too long to capture the stochastic intricacies among the arriving impressions. Hence, we tried to re-capture the same elements, the type graph and the Markov chain on a smaller snapshot of data. We constructed multiple data frames of about 6000 bidding starting from a random time for the log files. We sorted the bids in the bigger log file chronologically to preserve the order of impression arrival and re-ran the same experiment steps.

On the smaller log file, we found that the number of different advertisers and different types of impressions were generally limited to 3 and 6 respectively. Impressions of say, type $1, 2$ and 3 had only 1 advertiser interested in them, and impressions of type $4, 5$ and 6 had (a subset of) 2 advertisers interested in them. A large portion, 99.7% among the 6000 bids compromised of impressions of type $1, 2$ and 3. Again, the Markov chain contained scattered data which made it impossible to recognize patterns in its structure.

From above, we can infer that the underlying bipartite graph contains two types of impressions, (1) which has only 1 advertiser interested in it, and (2) an impression which has 2 set of advertisers interested in it. When type-(1) impressions occur in the input sequence, matching should be quite straightforward, the *only* interested advertiser matches with the impression. But in the case of impressions with 2 interested advertisers, we can't directly state any such results. Hence, we proposed working on such cases, in which online vertices have 2 choices of matching in the offline vertex set. This gave the formulation of the problem of finding matching in online bipartite graphs where the online vertices have degree 2, or 2 neighbors present in the offline vertex set. Naturally, any such online vertex having the same 2 neighbors in the offline set can essentially be abstracted to be the same type of vertices. This latent structure leads to the construction of a $(2, 2)$- regular

bipartite graph, which becomes the type graph upon which we present results.

Similarly, focusing on only the type-(2) impressions in the input sequence gave rise to an empirical Markov chain which had the structure of performing lazy random walk on a graph, with probability $\frac{1}{2}$. Once an impression arrived, it stayed in the same state (or it occurred again in the input sequence) with a probability of $\frac{1}{2}$, and switched to other states (or impressions) with a combined probability of $\frac{1}{2}$ distributed equivalently among the states. Two examples of such are presented as follows:

$$\begin{bmatrix} 0.500 & 0.417 & 0.083 \\ 0.313 & 0.562 & 0.125 \\ 0.500 & 0.250 & 0.250 \end{bmatrix}$$

(a) Markov chain based on 6000 bidding logs in chronological order from time 11 : 44 : 09.816 to 11 : 46 : 13.299 of date 12 June 2013 log file of the experiment data set.

$$\begin{bmatrix} 0.852 & 0.148 \\ 0.592 & 0.408 \end{bmatrix}$$

(b) Another Markov chain based on 6000 bidding logs in chronological order from time 23 : 30 : 05.367 to 23 : 30 : 07.503 of date 12 June 2013 log file of the experiment data set.

Figure 6.1: Example of empirical Markov chains derived from the experiment.

We generalized this $\frac{1}{2}$ probability lazy random walk to a general random walk of probability p remaining in the same state and remaining $1 - p$ probability of transitioning to other $n - 1$ states.

Chapter 7

Conclusion & Future Work

We wrap up this thesis by summarizing the results we attained on studying the online bipartite matching problem under the parameterized lazy random walk Markov chain.

We first summarized different input models for the problem of online bipartite matching and then introduced a new model for generating inputs based on Markov chain. This model helps in capturing the interaction between the online inputs, and this was the first time in the domain of online bipartite matching that dependency between input nodes was considered.

To study the problem, we began our research by looking into real world applications of the online matching problem from where we can analyze the structure of dependency among the online input items. We observed a good example in the online advertisement bidding experiment by iPinYou, a Chinese DSP where we looked deep into the data sets to understand how the experiment can be modeled into a bipartite graph matching problem. We extracted an underlying bipartite graph and used frequency analysis to create a Markov chain on the sequence of submitted bids.

We formulated the above empirical results into theoretical settings, in particular by studying the problem on the underlying extracted graph – a $(2, 2)$-regular bipartite graph, under a family of parameterized Markov chains resembling lazy random walks on a discrete state space. With parameter p depicting the probability of remaining in the same state and $\frac{1-p}{n-1}$ of entering a new state, we defined a Markov chain on the set of online vertices, which helped in generating the input sequence for the problem with Markovian dependency. We created an offline optimal algorithm, Frequency Based Optimal Matching, which is arguably a simpler algorithm to create an optimal

matching in offline settings. The run time of the algorithm was linear, and we analyzed the algorithm to obtain the expected number of matching in the input sequence generated by the parameterized Markov chain.

Next, we analyzed two versions of a popular algorithm in the online matching literature, Two Suggested Matching – Non Adaptive and Adaptive. We analyzed the performance of the algorithms under the input sequence sampled from the lazy random walk based Markov chain, and we obtained the expected size of matching in both cases. Coupled with the former results of the expected size of matching in the offline case, we derived the competitive ratio of the 2 algorithms. For specific case of the conventional lazy random walk Markov chain, where $p = \frac{1}{2}$, we found that the algorithm achieves competitive ratio of 0.9509 ($TSM - NA$) and 0.9733 ($TSM - A$), or matches at least the fraction of vertices as compared to OPT.

Because the family of Markov chains and the structure of the underlying type graph can be generalized vastly, one can go in varying directions in continuing the research from here on. We pose a few naturally occurring problems from our discussion so far.

(1) The work concerns $(2, 2)$-regular bipartite graphs, where the degree of each vertex is 2. This leads to 2 perfect and disjoint matchings. It can be noted that a (d, d)-regular bipartite graph can also be decomposed into d perfect disjoint matchings (the proof follows from application of Hall's marriage theorem and Konig's Edge Coloring theorem). Does the structure of analysis hold for a (d, d)-regular bipartite graph, albeit for a more general online algorithm than Two Suggested Matching? An approach for analyzing bipartite graphs with k perfect disjoint matchings is discussed in Karande et al. [48]. A more general question would be to find a matching in a (x, y)-regular bipartite graph or a more general type graph.

(2) An immediate follow up question from the discussion in Chapter 6 is to analyze the type graph where the vertices have degree at most 2. We analyzed the algorithm for $(2, 2)$-regular bipartite graph, where the degree of each vertex is exactly 2, but we suppose if the type graph has offline vertices with degree 1, then matching the vertex would simply constitute of picking the edge with an incoming online vertex. This shouldn't degrade the expected size of matching, as the offline vertex with degree 1 doesn't affect the matching of other vertices. Does the same hold for online vertices with degree 1? Another question would be to see how the Markov chain needs to be modified to

account for the degree 1 vertices, and then analyze the algorithm.

(3) A more general and important question would be to see if one can find some structure in the transition matrix of the Markov chain, using properties in the field of stochastic processes and spectral theory to see if it helps in the decision making process of matching a vertex. One can start by studying specific families of Markov chains like we did to find a correlation between some specific matrix properties e.g. eigenvalues and eigenvectors, the symmetric difference between 2 matrices, and the size of matching. A specific question that we focused on during the time of research was –

Given a Markov chain $\mathcal{M}_1(\pi_1, M_1)$ which attains a competitive ratio of ρ_1 on bipartite graph $G(U, V, E)$, then does another Markov chain $\mathcal{M}_2(\pi_2, M_2)$ with $\mathcal{M}_2 = f(\mathcal{M}_1)$ (e.g. a transformation of $\mathcal{M}_1$) attain a competitive ratio ρ_2, which is also a function of ρ_1?

Bibliography

[1] Atila Abdulkadiroğlu, Parag A. Pathak, and Alvin E. Roth. The New York city high school match. *American Economic Review*, 95(2):364–367, 2005.

[2] Gagan Aggarwal, Gagan Goel, Chandran Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proc. of 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 1253–1264. SIAM, 2011.

[3] Ahmed Al-Herz. *Approximation Algorithms for Maximum Vertex-Weighted Matching*. PhD thesis, Purdue University, 2019.

[4] Itai Ashlagi. Kidney exchange. In Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors, *Online and Matching-Based Market Design*, pages 201–216. Cambridge University Press, 2023.

[5] Itai Ashlagi and Alvin E. Roth. Kidney exchange: An operations perspective. *Management Science*, 67(9):5455–5478, 2021.

[6] Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *Proc. of 18th Annual European Symposium on Algorithms (ESA 2010)*, pages 170–181. Springer, 2010.

[7] Claude Berge. Two theorems in graph theory. *Proc. of the National Academy of Sciences of the United States of America*, 43(9):842–844, 1957.

[8] Benjamin Birnbaum and Claire Mathieu. Online bipartite matching made simple. In *Proc. of 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 530–537. SIAM, 2012.

[9] Norbert Blum. A new approach to maximum matching in general graphs. In *Proc. of 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, pages 586–597. Springer, 1990.

[10] Allan Borodin and Denis Pankratov. Online and other myopic algorithms. Manuscript.

[11] Niv Buchbinder, Kamal Jain, and Mohit Singh. Secretary problems via linear programming. *Mathematics of Operations Research*, 39(1):190–206, 2014.

[12] T.-H. H. Chan, Fei Chen, Xiaowei Wu, and Zhichao Zhao. Ranking on arbitrary graphs: Rematch via continuous lp with monotone and boundary condition constraints. In *Proc. of 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1112–1122. SIAM, 2014.

[13] Nikhil Devanur and Aranyak Mehta. Online matching in advertisement auctions. In Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors, *Online and Matching-Based Market Design*, pages 130–154. Cambridge University Press, 2023.

[14] Anatolii A. Dinic. Algorithm for solution of a problem of maximum flow in a network. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[15] Florin Dobrian, Mahantesh Halappanavar, Alex Pothen, and Ahmed Al-Herz. A 2/3-approximation algorithm for vertex weighted matching in bipartite graphs. *SIAM Journal on Scientific Computing*, 41(1):A566–A591, 2019.

[16] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[17] Andrew L. Dulmage and Nathan S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

[18] Federico Echenique, Nicole Immorlica, and Vijay V. Vazirani, editors. *Online and Matching-Based Market Design*. Cambridge University Press, 2023.

[19] Alon Eden, Michal Feldman, Amos Fiat, and Kineret Segal. An economics-based analysis of ranking for online bipartite matching. In *Proc. of 4th Symposium on Simplicity in Algorithms (SOSA 2021)*, pages 107–110. SIAM, 2021.

[20] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[21] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the Society for Industrial and Applied Mathematics*, 19(2):248–264, 1972.

[22] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[23] Jenő E. Egerváry. Matrixok kombinatorius tulajdonságairol (On the Combinatorial Properties of Matrices). *Matematikai Fizikai Lapok*, 38:16–28, 1931.

[24] Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. *Journal of the ACM (JACM)*, 69(6):1–35, 2022.

[25] Matthew Fahrbach and Morteza Zadimaghaddam. Online weighted matching: Breaking the 1/2 barrier. In *Proc. of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, pages 1242–1252. IEEE, 2019.

[26] Jon Feldman, Nitish Korula, Vahab Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Proc. of 5th International Workshop on Internet and Network Economics (WINE 2009)*, pages 374–385. Springer, 2009.

[27] Xixuan Feng. Online bipartite matching: a survey and a new problem. https://pages.cs.wisc.edu/~xfeng/sides/full_online.pdf, 2014. Accessed: 28-11-2024.

[28] Lester R. Ford Jr and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[29] Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[30] Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proc. of 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1990)*, page 434–443. SIAM, 1990.

[31] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM (JACM)*, 38(4):815–853, 1991.

[32] David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[33] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *Proc. of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, pages 26–37. IEEE, 2019.

[34] Caelan Garrett, Chris Graves, and Casey O'Brien. Survey on online bipartite matching and its variants. https://web.mit.edu/caelan/www/projects/6.854_paper.pdf. Accessed: 28-11-2024.

[35] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 982–991, 2008.

[36] Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *Proc. of the 7th International Workshop on Internet and Network Economics (WINE 2011)*, pages 170–181. Springer, 2011.

[37] Mahantesh Halappanavar. *Algorithms for Vertex-Weighted Matching in Graphs*. PhD thesis, Old Dominion University, 2009.

[38] John E. Hopcroft and Richard M. Karp. An $O(n^{5/2})$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. Previously announced at the 12th Annual Symposium on Switching and Automata Theory, 1971.

[39] Zhiyi Huang, Ning Kang, Zhihao G. Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *Journal of the ACM (JACM)*, 67(3):1–25, 2020.

[40] Zhiyi Huang and Xinkai Shu. Online stochastic matching, poisson arrivals, and the natural linear program. In *Proc. of the 53rd Annual ACM Symposium on Theory of Computing (STOC 2021)*, pages 682–693. ACM, 2021.

[41] Zhiyi Huang, Xinkai Shu, and Shuyi Yan. The power of multiple choices in online stochastic matching. In *Proc. of the 54th Annual ACM Symposium on Theory of Computing (STOC 2022)*, pages 91–103. ACM, 2022.

[42] Zhiyi Huang, Zhihao G. Tang, and David Wajc. Online matching: A brief survey. *ACM SIGecom Exchanges*, 22(1):135–158, 2024.

[43] Zhiyi Huang, Zhihao G. Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals. *ACM Transactions on Algorithms*, 15(3):1–15, 2019.

[44] Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4):577–595, 1985.

[45] Patrick Jaillet and Xin Lu. Online stochastic matching: New algorithms with better bounds. *Mathematics of Operations Research*, 39(3):624–646, 2014.

[46] Billy Jin and David P. Williamson. Improved analysis of ranking for online vertex-weighted bipartite matching in the random order model. In *Proc. of 18th International Conference on Web and Internet Economics (WINE 2022)*, pages 207–225. Springer, 2022.

[47] Donald B. Johnson. Priority queues with update and finding minimum spanning trees. *Information Processing Letters*, 4(3):53–57, 1975.

[48] Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proc. of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 587–596. ACM, 2011.

[49] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[50] Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000.

[51] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An online algorithm for a class of matching problems. In *Proc. of the 22nd Annual ACM Symposium on Theory of Computing (STOC 1990)*, pages 352–358. ACM, 1990.

[52] Alexander V. Karzanov. An exact estimate of an algorithm for finding a maximum flow, applied to the problem on representatives. *Problems in Cybernetics*, 5:66–70, 1973.

[53] Anup Kavathekar. Maximum matching. https://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/Kavathekar/Maximum_Matching.pdf. Lecture Notes for CS105, Winter 2005, Dartmouth College, Accessed: 28-11-2024.

[54] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *Proc. of 21st Annual European Symposium on Algorithms (ESA 2013)*, pages 589–600. Springer, 2013.

[55] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Proc. of 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, pages 508–520. Springer, 2009.

[56] László Kozma. Inequalities cheat sheet. https://www.lkozma.net/inequalities_cheat_sheet/ineq.pdf. Accessed: 28-11-2024.

[57] Erik Krohn and Kasturi Varadarajan. Private communication, 2007. Reported by Goel and Mehta in SODA 2008 [35].

[58] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[59] Ravi Kumar, Manish Purohit, Aaron Schild, Zoya Svitkina, and Erik Vee. Semi-Online Bipartite Matching. In *Proc. of 10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, pages 50:1–50:20. Schloss Dagstuhl, 2019.

[60] Dénes Kőnig. Gráfok és mátrixok. *Matematikai és Fizikai Lapok (Graphs and Matrices)*, 38:116–119, 1931.

[61] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proc. of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 597–606. ACM, 2011.

[62] Vahideh H. Manshadi, Shayan O. Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Mathematics of Operations Research*, 37(4):559–573, 2012.

[63] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4):265–368, 2013.

[64] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22–es, 2007.

[65] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proc. of 21st Annual Symposium on Foundations of Computer Science (FOCS 1980)*, pages 17–27. IEEE, 1980.

[66] Alvin E. Roth and Elliott Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *American Economic Review*, 89(4):748–780, 1999.

[67] Alvin E. Roth and Marilda Sotomayor. Two-sided matching. In *Handbook of Game Theory with Economic Applications*, volume 1, pages 485–541. Elsevier, 1992.

[68] Thomas H. Spencer and Ernst W. Mayr. Node weighted matching. In *Proc. of 11th International Colloquium on Automata, Languages and Programming (ICALP 1984)*, pages 454–464. Springer, 1984.

[69] Jules Tannery. *Introduction à la théorie des fonctions d'une variable*, volume 1. A. Hermann, 1910.

[70] Nobuaki Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.

[71] Vijay V. Vazirani. A simplification of the mv matching algorithm and its proof, 2013.

[72] Yajun Wang and Sam C. Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Proc. of 42nd International Colloquium on Automata, Languages and Programming (ICALP 2015)*, pages 1070–1081. Springer, 2015.

[73] Shuyi Yan. Edge-weighted online stochastic matching: Beating 1-$\frac{1}{e}$. In *Proc. of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2024)*, pages 4631–4640. SIAM, 2024.

[74] Real Time Advertiser Ad-slot Auction Data set. https://www.kaggle.com/datasets/saurav9786/real-time-advertisers-auction. Accessed: 28-11-2024.

[75] iPinYou Real Time Bidding Data set. https://www.kaggle.com/datasets/lastsummer/ipinyou. Accessed: 28-11-2024.

[76] Online eBay Auction Data set. https://www.kaggle.com/datasets/onlineauctions/online-auctions-dataset. Accessed: 28-11-2024.

[77] Patient Oxygen Demand Data set. https://www.kaggle.com/datasets/dibyasankhapal/realtime-patient-data-with-oxygen-demand. Accessed: 28-11-2024.

[78] Uber Customers Taxis Data Set. https://www.kaggle.com/datasets/anupammajhi/uber-request-data. Accessed: 28-11-2024.