Generating Randomized Maximum Bipartite Matchings Using Markov Chain Monte Carlo Via Swaps and Augmenting Paths

Amit Joshi
Email: amitjoshi24@utexas.edu

May 2022

1 Abstract

In this paper I explore generating random maximum bipartite matchings (MBPs) using Markov Chain Monte Carlo. I show that given an arbitrary MBP of a given graph, it is possible to generate a random MBP across the space of all MBPs of a graph. I explicitly define a Markov Chain with state space equal to the set of all MBPs to accomplish this, and prove correctness. I show that the MBPs generated via my Markov Chain follow a probability distribution that assigns positive probability to every MBP of a given graph. I show that this probability distribution is invariant under the initial MBP passed into my algorithm, and rather only depends on the graph itself (Graph-Dependent distribution), and that this distribution is the unique stationary one for this Markov Chain. I also show how to modify this Markov Chain so that the stationary distribution is uniform in order to generate uniformly random MBPs over a given graph, and prove that this is indeed the case, although this may take superpolynomial time to converge. Assuming certain conjectures hold true, I prove that my Graph-Dependent Markov Chain converges rapidly (polynomially) to its stationary distribution using Coupling.

2 Introduction

2.1 Related Work

Prior work has been done on generating a random maximum bipartite matching with high probability (Mitzenmacher and Upfal, 2005). They also used Glauber Dynamics and a coupling method to prove that their algorithm will generate a random maximum bipartite matching with high probability. Like me, they formulate their problem as a Markov Chain Monte Carlo (MCMC) and prove the mixing time (time it takes to generate a uniformly random solution) of their algorithm.

The work "On the switch Markov chain for perfect matchings" (Dyer et al., 2017) introduces the Switch Markov Chain, which "switches" the matches of a random pair of nodes as its transition, if possible. However, this Markov Chain is not *ergodic* across all graphs, meaning that it may not generate certain matchings if they are not reachable via these swaps.

There has been some work done on uniformly sampling *perfect* matchings, which is a maximum matching where every node is indeed matched. The work "Approximating the Permanent" (Jerrum and Sinclair) attempts to approximate the Permanent of a $n \times n$ matrix A with $0-1$ entries i and j, which is as follows:

$$Per(A) = \sum_{\sigma} \prod_{i=0}^{n-1} a_{i\sigma(i)}$$

Intuitively, this is the number of allowed permutations in a matrix A. The permanent function arises naturally in fields including algebra, combinatorial enumeration, and the physical sciences (Jerrum and Sinclair 1989). One such example is an application in statistical physics (Lyons) called the dimer model. Kasteleyn (Kasteleyn 1967) showed that for planar graphs $Per(A)$ can be computed in polynomial $O(n^3)$ time by constructing another matrix B where $det(B) = per(A)$, where det is the determinant. However, for general graphs, computing the permanent has been reduced to counting the number perfect bipartite matchings in the graph $G = (V, E)$ where $V = L \cup R$ where $L \cap R = \emptyset$ and $(i, j) \in E$ if and only if $a_{ij} = 1$. A paper by Broder (Broder 1988) reduces the problem of approximately counting number of perfect matchings to (almost) uniformly sampling them. This Markov Chain has a state space of all perfect and "near-perfect" (missing one edge) matchings. This work, and similar work, now must deal with the rather thorny problem of ensuring that this Markov Chain is rapidly mixing, meaning that within a polynomial number of transitions, the final state is virtually independent of the input (initial arbitrary perfect matching). In other words, after a relatively small (polynomial) number of transitions, the Markov Chain must arrive at a nearly uniformly random state. Broder (Broder 1988) used a complex (and unfortunately, incorrect) coupling argument (attempting to show that the expected difference between an arbitrary state and a uniformly random state decreases after applying the same transition operation to both) to show that his Markov Chain is rapidly mixing for dense graphs, where the minimum degree of a node is $\frac{n}{2}$. Fortunately, the paper by (Jerrum and Sinclair 1989) showed that Broder's Markov Chain is indeed rapidly mixing for dense graphs. Hence, they establish a fully polynomial randomized approximation scheme (fpras) for approximating the permanent, by successfully uniformly sampling from the space of perfect bipartite matchings. Later, Jerrum and Sinclair use this approach by Broder and themselves to show that an fpras exists for computing the number of perfect matchings for arbitrary dense graphs, and not just bipartite graphs. Importantly, they note that coupling is difficult for complex transitions where not a lot of symmetry is there, but conductance may work better in these types of Markov chains.

Another paper (Miklos and Kresz 2020) showed that counting maximum matchings (not necessarily perfect matchings) in planar graphs is P-complete, even though counting perfect matchings in planar graphs can be done in polynomial time. Note that this is just for graphs in general, not necessarily bipartite graphs.

However, to the best of my knowledge, there hasn't been any work done on taking an existing Maximum Bipartite Matching (MBP) and subsequently randomizing it, where the Markov Chain is guaranteed to lead to a MBP, and where every possible MBP has a positive probability of being generated. I show that this distribution is Graph-Dependent (GD), meaning that the distribution does not depend on the initial state (MBP, which nodes are matched to which other nodes) passed in, it only depends on the graph as a whole (the total set of vertices and edges). In this paper I explore an algorithm that will take an existing MBP (which can be computed via a variety of known

algorithms, such as Ford-Fulkerson, Hopcroft-Karp, Dinic's) and perform a series of operations that will take it asymptotically closer to a random MBP from the GD distribution.

2.2 Maximum Bipartite Matchings

A bipartite graph (V, E) is one in which the vertices of the graph (V) can be split into two sets L and R, and the only edges (E) in the graph are of the form $(l, r), l \in L, r \in R$ (Kleinberg and Tardos, 2014).

Alternatively, a bipartite graph can be thought of as satisfying the following constraints: $\forall l_1, l_2 \in L, l_1 \neq l_2$, $(l_1, l_2) \notin E$ AND $\forall r_1, r_2 \in R, r_1 \neq r_2$, $(r_1, r_2) \notin E$ (Kleinberg and Tardos, 2014) (Ibrahim 2020).

A bipartite matching is a set of edges such that $\forall v \in V$, at most 1 edge in the bipartite matching touches v (has v as an endpoint). A maximum bipartite matching is a bipartite matching that maximizes the number of vertices touching an edge. For matching i, let the edges in i be denoted as E^i.

For bipartite graphs, finding a MBP can be solved in polynomial time by using a max-flow algorithm. For example, the Ford-Fulkerson algorithm has runtime $O(|V||E|)$. Another algorithm, the Hopcroft-Karp algorithm in $O(|E|sqrt(|V|))$ time. Note that these are exact and deterministic algorithms (Kleinberg and Tardos, 2014).

Randomizing maximum bipartite matchings has many uses such as in assigning workers to tasks, assigning students to lessons, etc. For example, if each worker can only work on one task at once, and the next day if a random configuration (mapping from worker to task) is needed, the algorithm/results from this paper can be used.

One paper (Dyer et al., 2017) notes how sampling maximum matchings has applications in Statistics and Statistical Physics. Sampling MBPs from the uniform distribution can also be used to estimate the total number of MBPs of a graph.

For the readers convenience, I give an algorithm in the appendix for computing an arbitrary (but not uniformly random) maximum bipartite matching, known as the Augmenting Path Algorithm (Ibrahim 2020). This paper aims to take an arbitrary MBP and randomize it across the space of all MBPs of the graph.

2.2.1 Residual Graphs

Residual Graphs are directed graphs which are based upon the current matching, and are used in order to find an *augmenting path*, which, if found, will increase the number of edges in the current matching by 1 (Kleinberg and Tardos, 2014).

Definition 2.1 (Left Residual Graph)**.** A left residual graph $Res_L(i)$ on graph G and matching i can be found by doing the following (Kleinberg and Tardos, 2014):

1. Add all nodes in V to $Res_L(i)$
2. For each undirected edge of the form $(l, r) \in E$ where $l \in L$ and $r \in R$, if $(l, r) \in E^i$ then add the directed edge (r, l) to $Res_L(i)$.
3. For each undirected edge of the form $(l, r) \in E$ where $l \in L$ and $r \in R$, if $(l, r) \notin E^i$ then add the directed edge (l, r) to $Res_L(i)$.

Definition 2.2 (Right Residual Graph)**.** A right residual graph $Res_R(i)$ on graph G and matching i (any matching, does not have to be maximum bipartite matching) can be found by doing the following:

1. Add all nodes in V to $Res_R(i)$
2. For each undirected edge of the form $(l, r) \in E$ where $l \in L$ and $r \in R$, if $(l, r) \in E^i$ then add the directed edge (l, r) to $Res_R(i)$.
3. For each undirected edge of the form $(l, r) \in E$ where $l \in L$ and $r \in R$, if $(l, r) \notin E^i$ then add the directed edge (r, l) to $Res_R(i)$.

2.2.2 Randomized Augmenting Path Algorithm

I modify the augmenting path algorithm from (Ibrahim 2020) to randomized. One is a randomized left augmenting path, which takes in a left residual graph and tries to find a path from a (given) unmatched node in L to an unmatched node in R. However, the modification is that the order in which the out-neighbors of all $l \in L$ are processed. This is done in order to ensure that each possible augmenting path has some positive probability of being taken.

Algorithm 1 find-randomized-left-augmenting-path

```
1:  procedure RANDOMIZED-DFS(Res(G),node,visited,match)
2:      if node ∈ visited then
3:          return false
4:      end if
5:      add node to visited
6:      randomize order of neighbors of node
7:      for neighbor nb of node do
8:          if nb unmatched OR randomized − dfs(match[nb]) then
9:              match[node] ← nb
10:             match[nb] ← node                ▷ match node and nb to each other
11:             return true
12:         end if
13:     end for
14: end procedure
15: procedure FIND-RANDOMIZED-LEFT-AUGMENTING-PATH(G, a)   ▷ Parameters: Graph G,
    matching a
16:     shuffle − order ← random permutation of [1, 2, ..., |L|]
17:     visited ← ∅
18:     Res(G) ← left residual graph of G
19:     match ← a                    ▷ Mapping from nodes to their matches in a
```

```
    for unmatched l ∈ L ordered by shuffle − order do
        found ← dfs(Res(G), l, visited, match)
        if found = true then
            break
        end if
    end for
end procedure
procedure FIND-RANDOMIZED-RIGHT-AUGMENTING-PATH(G, a)      ▷ Parameters: Graph G, matching a
    shuffle − order ← random permutation of [1, 2, ..., |R|]
    visited ← ∅
    Res(G) ← right residual graph of G
    match ← a                                   ▷ Mapping from nodes to their matches in a
    for r ∈ R ordered by shuffle − order do
        found ← dfs(Res(G), r, visited, match)
        if found = true then
            break
        end if
    end for
end procedure
```

The other is a randomized right augmenting path which is symmetric and takes in a right residual graph and tries to find a path from an unmatched node in R to an unmatched node in L, similarly randomizing order that the out-neighbors (edges leaving $r \in R$) are processed.

Note that each left augmenting path starts with an unmatched node on the left, and tries to find a path in the left residual graph to an unmatched node on the right. Each directed edge in $Res_L(i)$ taken in the left augmenting path of the form (l, r) is added to i and each directed edge in $Res_L(i)$ taken in the left augmenting path of the form (r, l) is removed from i. As the left augmenting path starts from some unmatched $l \in L$ and ends in some unmatched $r \in R$, there is one more edge added than removed, so each augmenting path adds one more edge to a matching. WLOG, a right augmenting path works similarly.

2.3 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) algorithms have applications in statistics, physics, and computer science (Jindal et al., 2020). The idea behind MCMC is to design a Markov Chain whose stationary distribution is the one desired (Mitzenmacher and Upfal, 2005). In this paper, I give a Markov Chain whose stationary distribution is the uniform random distribution, as my aim is to generate a uniformly random MBP.

2.3.1 Markov Chains

Let Ω be the finite state space a Markov Chain operates over. Consider a stochastic process/sequence $(X_t)_{t=0}^{\infty}$. Conider the transition matrix P, of dimension $|\Omega| \times |\Omega|$. Note that:

$$\sum_{j \in \Omega} P_{i,j} = 1 \qquad \forall i \in \Omega$$

Process X is a Markov Chain if the probability that X transitions from i to j is independent of every transition prior to coming to i at that point. Mathematically, this is written as:

$$Pr[X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, ..., X_1 = x_1] = Pr[X_{t+1} = x_{t+1} | X_t = x_t]$$

Let's define some relevant terms to get a sense of the vocabulary.

- Stationary: A distribution π is called stationary if $P\pi = \pi$, meaning that the transition matrix does not affect the distribution.
- Irreducible: $\forall i, j \in \Omega$, $\exists t$ such that $P^t_{i,j} > 0$. In other words, an irreducible markov chain is one where every state can eventually get to another state with some positive probability.
- Aperiodic: $\forall i \in \Omega$, $gcd(t, P^t_{i,i}) = 1$. One way to show aperiodicity is the existence of self-loops in each state, meaning that with some positive probability a state i stays at itself after applying the transition.
- Time Reversibility: $\forall i, j$ we have that $\pi_i P_{ij} = \pi_j P_{ji}$. This means that the probability of being at i and transitioning from i to j is the same as the probability of probability of being at j and transitioning from j to i.

A Markov Chain has a *unique* stationary distribution (is ergodic) if it is irreducible and aperiodic by the Fundamental Theorem of Markov Chains (Mitzenmacher and Upfal, 2005).

2.3.2 Coupling

Coupling is a technique to estimate the expected difference between a state and a state from the stationary distribution in a Markov Chain (Mitzenmacher and Upfal, 2005). Consider the tuple of stochastic processes $(X, Y) \in \Omega \times \Omega$. Let Y be a random state from the stationary distribution (which is unknown to us, as that is what we are trying to approximate), and let X be the current known state. Let X_t, Y_t be these states at timestep t. Note that X_0 is the initial state before applying P at all. I attempt to perform the same transition to both X and Y (by applying the same transition matrix P to both), so note that:

- If $X_t = Y_t$ then $X_{t+1} = Y_{t+1}$

Define a distance function $d(X, Y)$ which computes the difference between two states in Ω. The aim of coupling is to decrease the *expected* distance between X and Y. If the expected distance between X and Y decreases, then the Markov Chain is said to *converge*.

2.3.3 Mixing Time

The mixing time is the time needed for expected difference between the states of the coupling to be small. Once the difference between X and Y is small ($\mathbb{E}[d(X, Y)] < 1$), I consider X to be converged to the stationary distribution (uniformly random). Mixing Time is found by proving expected bounds on how $d(X, Y)$ changes per each transition. A Markov Chain that converges can either be rapidly (polynomially) mixing or slow (superpolynomially) mixing.

3 Methods

3.1 Markov Chain for Graph-Dependent Distribution

Define the bipartite graph $G = (V, E)$ by splitting V split into two sets L and R, and the only edges (E) in the graph are of the form $(l, r), l \in L, r \in R$. There exists a maximum bipartite matching, which can be found via the Augmenting Path algorithm or by a reduction to max-flow and using a max-flow algorithm. Let m be the size of any MBP of G, which is the number of edges in any MBP of G.

Now, let's define the Markov Chain $M1$ and its transition matrix P. I will prove that this Markov Chain is irreducible and aperiodic, and that the uniform distribution is stationary under P. Let the finite state space Ω be the set of all possible maximum bipartite matchings for G. Let's define a state $i \in \Omega$, so i is a MBP. Let V^i be the vertices in i, and let E^i be the edges in i. Define $f_i : V \to V \cup \{-1\}$ to be the function that maps a vertex to its current match, or -1 if it is unmatched. Note that the match of a match of a vertex $v \in V$ is the vertex itself, $f_i(f_i(v)) = v$.

For any state $i \in \Omega$, where i is a MBP (Lemma 3), let the transition work as follows:

1. Select vertex $(v, w) \in E$ uniformly at random
2. If $(v, w) \in E^i$ do nothing, or if v is already matched to w (aka $f(v) = w$) then also do nothing
3. If $(v, w) \notin E^i$ match via the protocol below:
    - If w unmatched, simply match v to w by removing $(v, f_i(v))$ which allows for the addition of (v, w) as now both v and w are unmatched (right-swap transition).
    - If v unmatched, simply match v to w by adding (v, w) via removing $(f_i(w), w)$ (left-swap transition). Note that both v and w could not have been unmatched because then we could add edge (v, w) to obtain a larger matching than i, contradicting the fact that it is an MBP.
    - If neither are unmatched initially, do the following procedure. Consider the subgraph $G' = (V', E')$ which is the same as G but does not contain the nodes v and w and as a result does not contain any edges touching v or w (every other node/edge remains the same). Now i without v and w be known as $i(G')$, called here a "subgraph MBP." Now, attempt to find a randomized left augmenting path in $i(G')$, which is guaranteed to be found as long as (v, w) exists in any MBP of G by Lemma 1 (see Proof of Lemmas section). Note that these edges of the augmenting paths are ordered in such a way that it starts with some $l \in L$ and ends with some $r \in R$. Denote the result of finding an augmenting path in $i(G')$ as $i'(G')$. If an augmenting path is not found, then do nothing in this transition, as it means (v, w) can't be in any MBP of G by Lemma 1.

Theorem 1. *The above Markov Chain $M1$ is irreducible and aperiodic, and hence is ergodic and has a unique stationary distribution by the Fundamental Theorem of Markov Chains.*

Proof. First, I will show that this Markov Chain is irreducible. Let i and j be two MBPs. To show irreducibility need to show that with some positive probability it is possible to go from i to j under some sequence of transitions above. Define $G^{ij} = (V, E^i \cup E^j - \{E^i \cap E^j\})$. These edges will lead G^{ij} to be a collection of disjoint cycles and simple paths (Lemma 4). I disregard the edges that appear in both E^i and E^j because there is no need for a transition to change these edges. To get

from one state to the other, I will consider each cycle/path individually, and show that it is possible with some positive probability to transition from the cycle/path in i to its corresponding cycle/path in j (same nodes with different edges). Once all cycles/paths have been transitioned from i to j with some positive probability, we have transitioned completely from i to j.

In either i or j, the maximum degree of any node is 1 since a node can only be matched once. Hence, in the union of both matchings i and j, the maximum degree of any node is 2. Let an edge "border" another edge if they share a node as one of their endpoints. Then, note that an edge from the same matching can not border another edge from the same matching because that would mean that under a matching, one node is matched with two nodes. Hence, in any of the disjoint cycles/paths, each edge will only border edges that come from the opposite matching. For example, if $e^i \in E^i, e^i \notin E^j$, e^i will not border any different $e^{i'} \in E^i$.

Now, to transition a left-path of the form $l_1, r_1, l_2, r_2, \ldots, l_n$ (Lemma 7) from i to j do the following:

- As indexing in L and R is arbitrary, we can move nodes around in each column to explicitly define f_i, f_j. Keep in mind that the path $l_1, r_1, l_2, r_2, \ldots, l_n$ has no inherent order, we can just reverse it if necessary to let l_1 be the the node on the left coming from matching j, and hence unmatched in matching i.

$$f_i(l_i) = \begin{cases} r_{i-1} & i \neq 1 \\ -1 & i = 1 \\ & l_1 \text{ is unmatched in } i \end{cases} \tag{1}$$

$$f_j(l_i) = \begin{cases} r_i & i \leq n-1 \\ r_1 & i = n \end{cases} \tag{2}$$

  Hence, we know that r_1 is matched to l_1 in matching j (and hence not in i since we disregarded edges appearing in both matchings). Since l_1 had degree 1 in G^{ij}, and was connected to an edge in j, we know that l_1 is unmatched in i. We know that r_1 must actually be matched in i, since if r_1 was unmatched then an extra edge could be drawn from l_1 to r_1 in matching i, contradicting the fact that i is a MBP. So, we can perform a left swap on matching i by picking $(v, w) \in E$ where $v = l_1$ and $w = r_2$, which happens with nonzero ($\frac{1}{|E|}$) probability. Now, $f_i(l_1) = f_j(l_1) = r_1$. Now, note that we are still left with a path with two nodes of degree 1 on the left, since we no longer need to transition the matches of l_1 and r_1. So now our path becomes a subpath of the original path (and hence is still disjoint from the other paths/cycles in G^{ij}) becomes $l_2, r_2, \ldots, l_n$. Note that since we performed a left swap transition, we unmatched $f_i(r_1) = l_2$ in i, so we know that l_2 is unmatched in i. Since edges in the original path have to alternate as proved earlier, we know that edges of the form (l_i, r_i) came from E^j, so we know that $f_j(l_2) = r_2$. So, we can perform a left swap on matching i where $v = l_2$ and $w = r_2$, which happens with nonzero ($\frac{1}{|E|}$) probability. Now, $f_i(l_2) = f_j(l_2) = r_2$. Now, l_3 has degree 1, is unmatched in i, and matched in j, and we can perform another left swap. Repeat this process until $f_i(l_{n-1}) = r_{n-1}$. Note that l_n has degree 1 in G^{ij}, and came from E^i since l_1 came from E^j. Hence, l_n is unmatched in j and we should perform these left swaps $n-1$ times. All these left swaps occur with nonzero probability, namely:

$$Pr[\text{left\_swap}(v = l_i, w = r_i)] = \frac{1}{|E|} \tag{3}$$

$$\Pi_{i=1}^{n-1} Pr[\text{left\_swap}(v = l_i, w = r_i)] = \left(\frac{1}{|E|}\right)^{n-1} \tag{4}$$

Since all the transitions are independent

Transitioning a right-path of the form $r_1, l_1, r_2, l_2, \ldots, r_n$ (Lemma 7) from i to j is similar:

- As indexing in L and R is arbitrary, we can move nodes around in each column to explicitly define f_i, f_j. Keep in mind that the path $r_1, l_1, r_2, l_2, \ldots, r_n$ has no inherent order, we can just reverse it if necessary to let r_1 be the the node on the left coming from matching j, and hence unmatched in matching i.

$$f_i(r_i) = \begin{cases} l_{i-1} & i \neq 1 \\ -1 & i = 1 \\ & r_1 \text{ is unmatched in } i \end{cases} \tag{5}$$

$$f_j(r_i) = \begin{cases} l_i & i \leq n-1 \\ l_1 & i = n \end{cases} \tag{6}$$

As the path $r_1, l_1, r_2, l_2, \ldots, r_n$ has no inherent order, we can just reverse it if necessary to let r_1 be the the node on the right coming from matching j. Hence, we know that l_1 is matched to r_1 in matching j (and hence not in i since we disregarded edges appearing in both matchings). Since r_1 had degree 1 in G^{ij}, and was connected to an edge in j, we know that r_1 is unmatched in i. We know that l_1 must actually be matched in i, since if l_1 was unmatched then an extra edge could be drawn from r_1 to l_1 in matching i, contradicting the fact that i is a MBP. So, we can perform a right swap on matching i where $v = l_2$ and $w = r_1$, which happens with nonzero ($\frac{1}{|E|}$) probability. Now, $f_i(r_1) = f_j(r_1) = l_1$. Now, note that we are still left with a path with two nodes of degree 1 on the left, since we no longer need to transition the matches of l_1 and r_1. So now our path becomes a subpath of the original path (and hence is still disjoint from the other paths/cycles in G^{ij}) becomes $r_2, l_2, \ldots, r_n$. Note that since we performed a left swap transition, we unmatched $f_i(l_1) = r_2$ in i, so we know that r_2 is unmatched in i. Since edges in the original path have to alternate as proved earlier, we know that edges of the form (r_i, l_i) came from E^j, so we know that $f_j(r_2) = l_2$. So, now we can perform a right swap on matching i where $v = l_2$ and $w = r_2$, which happens with nonzero ($\frac{1}{|E|}$) probability. Now, $f_i(r_2) = f_j(r_2) = l_2$. Now, r_3 has degree 1, is unmatched in i, and matched in j, and we can perform another right swap. Repeat this process until $f_i(r_{n-1}) = l_{n-1}$. Note that r_n has degree 1 in G^{ij}, and came from E^i since r_1 came from E^j. Hence, r_n is unmatched in j and we should perform these right swaps $n-1$ times. All these right swaps occur with nonzero probability, namely:

$$Pr[\text{right\_swap}(v = l_i, w = r_i)] = \frac{1}{|E|} \tag{7}$$

$$\Pi_{i=1}^{n-1} Pr[\text{right\_swap}(v = l_i, w = r_i)] = \left(\frac{1}{|E|}\right)^{n-1} \tag{8}$$

Since all the transitions are independent

The last case to consider is transitioning a cycle from i to it's counterpart in j. Define the cycle as follows: $r_1, l_1, r_2, l_2, \ldots l_n$ where n nodes are in R and n nodes are in L (by Lemma 8).

- As indexing in L and R, is arbitrary, we can move nodes around in each column to let:

$$f_i(l_i) = r_i \tag{9}$$

$$f_j(l_i) = \begin{cases} r_{i+1} & i \leq n-1 \\ r_1 & i = n \end{cases} \tag{10}$$

Note that each pair of adjacent nodes in this cycle represents an undirected edge in G^{ij}. Specifically, edges of the form (r_i, l_i) come from matching i (since $f_i(l_i) = r_i$ and $f_i(r_i) = l_i$) and that edges of the form (l_i, r_{i+1}) and (l_n, r_1) come from matching j given the above definition of f_j.

Now, given that my augmenting path algorithm randomizes the order of neighbors of each vertex $v \in L$, each neighbor has a positive probability of being chosen for the augmenting path. For this to happen, $f_j(l)$ needs to be chosen as the edge to take out of l, as this means if an augmenting path is found, l will be matched to $f_j(l)$. Note that $f_j(l_n) = r_1$, which is unmatched before an augmenting path is attempted to be found. For this to happen for the vertices in L, wlog $v_1, v_3, \ldots v_{k-1}$, we can lower bound this probability by:

$$\Pi_{i'=0}^{n-1} \frac{1}{\delta(v_{2i'+1})} \qquad \delta(v) \text{ is out-degree of } v \text{ in } Res_L(i(G')) \tag{11}$$

Note that this probability is positive, so with positive probability the augmenting path found will transition the cycle from it's form in i to it's counterpart in j.

Once all of the vertices of all the disjoint cycles/paths are transitioned from their edges in i to their edges in j, then the state i has transitioned to j. Remember that those edges that appeared in both i and j did not have to transition, and they were not included in the disjoint cycles/paths.

Hence, it is possible to get from any MBP to any other MBP with some positive probability, and hence this Markov Chain is irreducible.

Next, it is easy to see that this Markov Chain is aperiodic due to the existence of self-loops (an MBP transitioning to itself). Note that in the transition, if $(v, w) \in E^i$ is randomly selected the MBP will stay the same (self-loop on i). Or, if an augmenting path is not found, the MBP will stay the same.

Hence, by the Fundamental Theorem of Markov Chains, this Markov Chain $M1$ is ergodic and has a unique stationary distribution. As this Markov Chain is over all MBPs, the initial MBP does not matter, only the graph itself matters, since no matter the inital MBP this Markov Chain will converge to a unique stationary distribution. So, denote this stationary distribution as the Graph-Dependent (GD) distribution, which gives positive probability to every MBP of G.

□

3.2 Markov Chain for Uniform Distribution

We can modify this Markov Chain $M1$'s transition matrix P so that it converges to the uniformly random distribution, where each MBP has equal probability of being generated. To do this, let's define a time-reversible (for the uniform distribution) Markov Chain, called $M2$. Let's denote it's transition matrix as $\hat{P}$. $\hat{P}$ is similar to P, but with some added self loops. In particular, it adds self-loops to ensure that $\hat{P}_{ij} = \hat{P}_{ji}$. This is done by setting $\hat{P}_{ij} = \hat{P}_{ji} = min(P_{ij}, P_{ji})$.

- If j is accessible from i via a swap transition:

  Without loss of generality, let's say that the swap transition was a right swap, so matching i has $(l, f_i(l))$ and matching j instead has $(l, f_j(l))$. For the transition from i to j to happen, the (v, w) chosen in step 1 must be $(l, f_j(l))$ which occurs with $\frac{1}{|E|} = P_{ij}$ probability. For the transition from j to i to happen, the (v, w) chosen in step 1 must be $(l, f_i(l))$ which also occurs with $\frac{1}{|E|} = P_{ji}$ probability. Hence, we have that $P_{ij} = P_{ji} = \hat{P}_{ij} = \hat{P}_{ji}$.

- If j is accessible from i via an augmenting path transition:

  To find the probability with which an augmenting path was taken, consider modifying the randomized augmenting path algorithm from before to not only select a random augmenting path but also calculate the probability with which it was chosen. Let's call this find-randomized-augmenting-path-prob algorithm. Let's use the find-randomized-left-augmenting-path to start, and then modify it. To do this, at each "randomized" edge taken, we will calculate the probability with which that edge would actually be taken with the same graph. Let's introduce the notion of viable edges. A viable edge is one where if that edge was taken, then an augmenting path would continue to be found. We can verify whether or not an augmenting path would continue to be found by using DFS (which is $O(|E|)$ to see if there is any path from the result of taking that edge to a goal state (unmatched node on the right). Let $\delta(v)$ represent the total possible edges out of node v conditioned on the augmenting path taken thus far. Let's say that there are $q(v)$ viable edges out of node v conditioned on the augmenting path taken thus far. To calculate $q(v)$, conduct $\delta(v)$ bread-first-searches to see which of these are viable. The augmenting path can only have a node at most once, so now the total time complexity of finding and augmenting path and calculating its probability is $O(|V||E|^2)$. Note that $q(v) \leq \delta(v)$. The probability that any one of these $q(v)$ viable edges was the first one considered is $\frac{1}{q(v)}$, as the non-viable edges can appear in any order and not affect the outcome of which edge is taken in the augmenting path. The first edge f of these $q(v)$ is the one that is taken. As the edges are randomly permuted, the probability that f is the first edge is $\frac{1}{q(v)}$. As the augmenting path continues to be found, multiply these probabilities together. If this augmenting path ends up being a cycle in G^{ij}, then there are multiple augmenting path transitions to accomplish this, each of which selects a distinct (v, w) in step 1. Adding all these probabilities together finds the true P_{ij}, but it is sufficient to simply set the probability that the y^{th} augmenting path transition from i to j ($\hat{P}^y_{ij}$) is equal to its opposite augmenting path transition from j to i ($\hat{P}^y_{ji}$), as if there are z augmenting path transitions from i to j there are z opposite augmenting path transitions (by Lemma 9), and if this transition represents a cycle in G^{ij} then $z > 1$. So, if $\hat{P}^y_{ij} = \hat{P}^y_{ji} = min(\hat{P}^y_{ij}, \hat{P}^y_{ji})$ then: $\hat{P}_{ij} = \sum_{y=1}^{z} \hat{P}^y_{ij} = \sum_{y=1}^{z} \hat{P}^y_{ji} = \hat{P}_{ji}$ as desired.

Algorithm 2 find-randomized-augmenting-path-prob

procedure RANDOMIZED-DFS-PROB($Res(G)$,$node$,$visited$,$match$, $curProb$)
  if $node \in visited$ **then**
    return $false$
  end if
  add $node$ to $visited$
  randomize order of neighbors of $node$
  $q(node) \leftarrow 0$
  $firstViable \leftarrow -1$
  for neighbor nb of $node$ **do**
    if nb unmatched **then**
      $q(node) \leftarrow q(node) + 1$
      if $firstViable = -1$ **then**
        $firstViable \leftarrow node$
      end if
      continue
    end if
    $visitedCopy \leftarrow deepcopy(visited) \cup \{node, nb\}$
    if $randomized - dfs(match[nb], visitedCopy)$ **then**
      $q(node) \leftarrow q(node) + 1$
      $firstViable \leftarrow node$
    end if
  end for
  if $firstViable = -1$ **then** ▷ no viable edges
    return $false$
  end if
  $nb \leftarrow firstViable$
  if nb unmatched **then**
    $match[node] \leftarrow nb$
    $match[nb] \leftarrow node$ ▷ match $node$ and nb to each other
    return $true, curProb$
  end if
  return $randomized - dfs(match[nb], visited, curProb/q(node))$
end procedure
procedure FIND-RANDOMIZED-AUGMENTING-PATH-PROB(G, a) ▷ Parameters: Graph G, matching a
  $shuffle - order \leftarrow$ random permutation of $[1, 2, \ldots, |L|]$
  $visited \leftarrow \emptyset$
  $Res(G) \leftarrow$ left residual graph of G
  $match \leftarrow a$ ▷ Mapping from nodes to their matches in a
  $q \leftarrow 0$ ▷ count viable nodes to start from
  $firstViable \leftarrow -1$
  for $l \in L$ ordered by $shuffle - order$ **do**
    if l unmatched **then**
      $found \leftarrow randomized - dfs(Res(G), l, deepcopy(visited), match)$
      if $found = true$ **then**
        $q \leftarrow q + 1$
        if $firstViable = -1$ **then**
          $firstViable \leftarrow l$
        end if
      end if
    end if
  end for

52: **if** $firstViable = -1$ **then**
53: return $false$
54: **end if**
55: return $randomized - dfs - prob(Res(G), l, visited, match, curProb = 1/q)$
56: **end procedure**

Now, to calculate P_{ji}, consider the algorithm calc-augmenting-path-prob. This algorithm is similar to find-randomized-augmenting-path-prob but instead it is given an augmenting path that it calculates the probability of. First, consider the case where this augmenting path transition represents a path in G^{ij}. Given the augmenting path transition that selects a (v, w) and then performs augmenting path say (WLOG) $l_1, r_1, l_2, r_2, \ldots r_n$, calculate the reverse augmenting path and calculate it's probability. Let the reverse augmenting path be $l'_1, r'_1, l'_2, r'_2, \ldots l'_n, r'_n$. For example, say l'_k has $\frac{1}{q(l'_k)}$ probability of going to r'_k (by calculating the number of viable edges of l'_k). Multiply all these probabilities together to find P_{ji}. For the case where this augmenting path ends up being a cycle in G^{ij}, then there are multiple augmenting paths to accomplish this, each of which selects a distinct (v, w) in step 1. So, there are actually z augmenting path transitions from j to i. As mentioned earlier, we can see that if $\hat{P}^y_{ij} = \hat{P}^y_{ji} = min(\hat{P}^y_{ij} = \hat{P}^y_{ji})$ for $y \in \{1, 2, ..., z\}$ then $\hat{P}_{ij} = \hat{P}_{ji}$.

We can see that P_{ij} may differ from P_{ji} but only in the case where j is accessible via an augmenting path transition due to the degrees of the vertices and the order in which the vertices are traversed. After setting $\hat{P}_{ij} = \hat{P}_{ji} = min(P_{ij}, P_{ji})$, one way to modify the transition defined for $\hat{P}$ is as follows. Once the augmenting path for transitioning from i is found to arrive at j, and the probability (P_{ij}) has been calculated via the find-randomized-augmenting-path-prob algorithm (which finds a randomized augmenting path and returns it and the probability with which it was taken), calculate P_{ji} using the calc-augmenting-path-prob which takes in an augmenting path and calculates the probability with which it is taken.
Case that $P_{ij} \leq P_{ji}$: transition to j. Hence we have that $\hat{P}_{ij} = P_{ij} = min(P_{ij}, P_{ji})$.
Case that $P_{ij} > P_{ji}$, since $\hat{P}_{ij}$ should be equal to $\hat{P}_{ji}$, after finding j, transition to j with probability $p = \frac{P_{ji}}{P_{ij}}$, and stay at i with probability $1 - p$ (in addition to the existing self-loops at i defined in P). Hence, we have that $\hat{P}_{ij} = P_{ij} * p = P_{ij} * \frac{P_{ji}}{P_{ij}} = P_{ji} = min(P_{ij}, P_{ji})$.

Algorithm 3 calc-augmenting-path-prob

1: **procedure** CALC-AUGMENTING-PATH-PROB-DFS($Res(G)$,$visited$,$match$,$curProb$,$path$,$pathIndex$)
2: $node \leftarrow path[pathIndex]$
3: **if** $node \in visited$ **then**
4: return $false$
5: **end if**
6: add $node$ to $visited$

```
    q(node) ← 0
    for neighbor nb of node do
        if nb unmatched then
            q(node) ← q(node) + 1
            continue
        end if
        visitedCopy ← deepcopy(visited) ∪ {node, nb}
        if randomized − dfs(match[nb], visitedCopy) then
            q(node) ← q(node) + 1
        end if
    end for
    nb ← path[pathIndex + 1]
    if nb unmatched then
        match[node] ← nb
        match[nb] ← node                          ▷ match node and nb to each other
        return true, curProb
    end if
    return randomized − dfs(match[nb], visited, curProb/q(node), path, pathIndex + 2)
end procedure
procedure CALC-AUGMENTING-PATH-PROB(G, a, path)  ▷ Parameters: Graph G, matching a, path path
    shuffle − order ← random permutation of [1, 2, ..., |L|]
    visited ← ∅
    Res(G) ← left residual graph of G
    match ← a                                    ▷ Mapping from nodes to their matches in a
    q ← 0                                        ▷ count viable nodes to start from
    for l ∈ L ordered by shuffle − order do
        if l unmatched then
            found ← randomized − dfs(Res(G), l, deepcopy(visited), match)
            if found = true then
                q ← q + 1
            end if
        end if
    end for
    l ← path[0]
    return calc − augmenting − path − prob − dfs(Res(G), l, visited, match, curProb = 1/q, path, pathIndex = 0)
end procedure
```

Theorem 2. *The uniformly random distribution is stationary for $M2$ and $\hat{P}$, and $M2$ will eventually converge to a uniformly random state.*

Proof. To see that the uniformly random distribution is stationary, consider any state i in Ω (consider any MBP of G) and consider all the states that are accessible via one transition.

Given that i comes from a uniformly random distribution, $\pi_i = \frac{1}{|\Omega|}$.

Let j be accessible from i via one transition. Since j also comes from the uniformly random

distribution, $\pi_j = \frac{1}{|\Omega|}$.

Note that j can potentially transition to i by selecting v and matching it to $f_i(v)$. By definition, this occurs with probability $\hat{P}_{ji}$.

So, for every probability of leaving state i to go to another state j under transition P, we have equal probability of coming to state i from state j under transition $\hat{P}$. Mathematically, we have that $\hat{P}_{ij} = \hat{P}_{ji}$. As $\pi_i = \pi_j$, we have that $\hat{P}$ is also time-reversible.

Consider the definition of a stationary distribution, where the probability of transitioning to any state i is the same as the probability of leaving any state i.

$$
\begin{aligned}
&Pr[\text{leaving } i] && && (12)\\
&= \sum_{\forall j \text{ reachable from } i} \pi_i \hat{P}_{ij} && && (13)\\
&= \sum_{\forall j \text{ reachable from } i} \pi_j \hat{P}_{ij} && \text{Since } \pi_i = \pi_j \text{ since uniform} && (14)\\
&= \sum_{\forall j \text{ reachable from } i} \pi_j \hat{P}_{ji} && \text{By construction of } \hat{P} && (15)\\
&= Pr[\text{coming to } i] && && (16)
\end{aligned}
$$

Hence, we have that:

$$\hat{P}\pi = \pi \qquad \text{where } \pi \text{ is the stationary distribution in this case} \qquad (17)$$

Hence, the uniformly random distribution is stationary. By the same argument made in Theorem 1, there is only one stationary distribution for $M2$, I just proved that it is the uniformly random distribution, so eventually, $M2$ will arrive at (or converge to) a uniformly random state. □

4 Markov Chain Monte Carlo Coupling

Let's now define the coupling (X, Y). X is an arbitrary maximum bipartite matching of G. Y is drawn from the stationary distribution of the Markov Chain $M1$, known in this paper as the Graph-Dependent (GD) distribution. Y is a random maximum bipartite matching, that we do not know, it is a state that exists in theory. Note that applying the transition matrix to Y still gives a Graph-Dependent (GD) random state, as Theorem 4 showed that the GD distribution is the unique stationary distribution. The aim of this coupling is to prove bounds on how long X will take to converge to this unique stationary distribution (GD), and hopefully show that it converges rapidly (in polynomial time). We have proved earlier that both P and $\hat{P}$ do converge eventually to their respective stationary distributions (although this may take superpolynomial time). Assuming two conjectures hold true, let's prove that at least P converges rapidly (polynomially) to its stationary distribution (GD distribution).

Realize that maximum bipartite matchings can be thought of as functions from a vertex to its matching vertex. So, define $f_X : V \to V \cup \{-1\}$ where $f_X(v) = f_X(w)$ if $(v, w) \in E^X$. Note that $f_X(v) = -1$ if v is unmatched. Similarly, define f_Y.

Let's define a distance function for this coupling d (distance from the left). Define $d(X, Y)$ to be the number of edges $e \in E$ that are in X but not Y. Note that d is symmetric in that

$d(X,Y) = d(Y,X)$ (Lemma 10). We consider X to be part of the stationary distribution when $d(X,Y) < 1$.

Theorem 3. *The mixing time (T_{mix}) of this coupling of $M1$ is: $\frac{|E|}{d} ln(|V|/2)$. This implies that the mixing time of the Markov Chain $M1$ is upper bounded by T_{mix}*

Proof. To see why this is, consider the expected change in $d(X,Y)$ after a transition. For readability, let $d = d(X,Y)$ Denote the result of applying transition matrix P to X and Y as X' and Y' respectively.

As stated before, let (v,w) be the edge chosen in step 1.

Consider the case where the (v,w) appears in both X and Y. I claim that if (v,w) appears in both X and Y, then $d(X',Y') = d(X,Y)$. This is a self-loop for both X and Y, and the transition P does not modify X or Y. Hence, $X' = X$ and $Y' = Y$ and hence $d(X',Y') = d = d(X,Y)$.

Now, let's calculate the probability that $d(X,Y)$ will decrease. I conjecture this to happen if the (v,w) chosen in step 1 appears in one of X or Y (X xor Y), but not both. First, note that the probability with which such an edge is chosen is $2d/|E|$, since d of those edges are in X and another d are in Y.

$$Pr[(v,w) \text{ being in } X \text{ xor } Y] = \frac{2d}{|E|} \tag{18}$$

Conjecture 3.1. If (v,w) appears in X xor Y, then applying P to both X and Y will decrease the distance between X and Y by at least 1. In other words, $\mathbb{E}[d(X',Y')] \leq \mathbb{E}[d(X,Y) - 1]$.

Reasoning for conjecture: Without loss of generality, say that (v,w) appears in X, but not Y. First, consider the case where (v,w) is added to Y via a swap transition. If it was a left swap, then $(f_Y(w),w)$ was removed from Y to allow for (v,w) in Y'. We know that $(f_Y(w),w)$ was not in X since it would have conflicted with $(v,w) \in E^X$ since w can only be matched to one node in X. If (v,w) was added via a right swap, then $(v,f_Y(v))$ was removed from Y to allow for (v,w) in Y'. We know that $(v,f_Y(v))$ was not in X since it would have conflicted with $(v,w) \in E^X$ since v can only be matched to one node in X. So, in either case one uncommon edge was removed from Y and replaced with a common edge to make Y', and X' stayed the same, so $d(X',Y') = d(X,Y) - 1$.

Next, consider the case where (v,w) is added via an augmenting path transition to Y. I conjecture that in expectation, this will decrease the distance by 1. Since (v,w) will now be common in X and Y by Theorem 1. I conjecture that a random augmenting path in expectation will make it such that $\mathbb{E}[d(X',Y')] \leq \mathbb{E}[d(X,Y) - 1]$.

I can show that $(v,f_Y(v))$ and $(f_Y(w),w)$ were not in X, so they can be removed from Y without increasing distance to X. Note that since $(v,w) \notin E^Y$ we know that $f_Y(v) \neq w$ and $f_Y(w) \neq v$. Knowing that $(v,w) \in X$, we know that this bars $(v,f_Y(v))$ and $(f_Y(w),w)$ from being in X since $(v,w) \in E^X$ means that both v and w were matched to each other in X. So, removing these edges from Y, does not increase its distance to X. Now, (v,w) is common to X and Y so in doing so the distance between X and Y decreased by 1.

I can conjecture that a random augmenting path in expectation, will not delete more edges in common than it will add.

Conjecture 3.2. I conjecture that if (v, w) appears in neither X nor Y, then in expectation the distance $d(X, Y)$ will not increase, so that $\mathbb{E}[d(X', Y')] \leq \mathbb{E}[d(X, Y)]$

> Reasoning for conjecture. Let's say that (v, w) is added via a swap transition in X and via an augmenting path transition in Y. WLOG the case where it is an augmenting path transition in X and a swap transition in Y works similarly.
>
> Consider X' which is X after the swap is performed. As a swap transition simply removes one edge and adds another edge, we know that $d(X', X) = 1$. Now, X' contains (v, w). So this reduces to the case where (v, w) is in one matcing and (v, w) is added via an augmenting path transition in the other. So, by Conjecture 3.1, we know that $d(X', Y') \leq d(X', Y) - 1$.

$$d(X', X) = 1 \qquad \text{By swap transition} \tag{19}$$
$$d(X', Y') \leq d(X', Y) - 1 \qquad \text{By Conjecture 3.1} \tag{20}$$
$$d(X', Y) \leq d(X', X) + d(X, Y) \qquad \text{By triangle inequality of } d\text{, Lemma 11} \tag{21}$$
$$d(X', Y) \leq 1 + d(X, Y) \qquad \text{By triangle inequality} \tag{22}$$
$$d(X', Y') \leq 1 + d(X, Y) - 1 = d(X, Y) \qquad \text{Combining (36) and (38)} \tag{23}$$

> Now for the other case where both are added via an augmenting path, I can conjecture that a random augmenting path in expectation, will not delete more edges in common than it will add, and that in expectation, at least one more common edge will be added so that even if two common edges were removed from X and Y in the transition, we can still have that $\mathbb{E}[d(X', Y')] \leq \mathbb{E}[d(X, Y)]$.

Assuming conjectures 3.1 and 3.2 hold true, let's calculate the mixing time of Markov Chain P. If these conjectures are true, we have that:

$$\mathbb{E}[d(X', Y')] = Pr[(v, w) \in X \text{ xor } Y] * (\text{expected change if } (v, w) \in X \text{ xor } Y) \tag{24}$$
$$+ Pr[(v, w) \in X \text{ and } Y] * (\text{expected change if } (v, w) \in X \text{ and } Y) \tag{25}$$
$$+ Pr[(v, w) \notin X, Y] * (\text{expected change if } (v, w) \notin X, Y) \tag{26}$$
$$= \frac{2d}{|E|} * (-1) + 0 + 0 \qquad \text{since latter two terms are 0} \tag{27}$$
$$= -\frac{2d}{|E|} \tag{28}$$

Hence, the expected *change* in $d(X, Y)$ is at most $-\frac{2d}{|E|}$ (meaning that the expected change is either more negative that that or the same).

$$\mathbb{E}[d_L(X^{t+1}, Y^{t+1})] \leq \mathbb{E}[d(X^t, Y^t)] \left(1 - \frac{2d}{|E|}\right) \tag{29}$$

$$\mathbb{E}[d_L(X^t, Y^t)] \leq \mathbb{E}[d(X^0, Y^0)] \left(1 - \frac{2d}{|E|}\right)^t \tag{30}$$

$$\leq m \left(1 - \frac{2d}{|E|}\right)^t \qquad \text{In the case that every vertex is matched differently} \tag{31}$$

$$\leq \frac{|V|}{2} \left(1 - \frac{2d}{|E|}\right)^t \qquad m \text{ can be upper bounded by } \frac{|V|}{2} \tag{32}$$

From equation (47), setting $t = 2\frac{|E|}{2d} ln(|V|/2) = T_{mix}$ gives us:

$$\mathbb{E}[d(X^t, Y^t)] \leq \frac{|V|}{2} \left(1 - \frac{2d}{|E|}\right)^t \tag{33}$$

$$\leq \frac{|V|}{2} \left(\frac{1}{e}\right)^{2ln(|V|/2)} \tag{34}$$

$$\leq \frac{|V|}{2} \left(\frac{1}{(|V|/2)^2}\right) \tag{35}$$

$$= \frac{2}{|V|} \leq 1 \qquad |V| \geq 2 \text{ for a nonempty matching} \tag{36}$$

Since after this many iterations of the transition, an arbitrary MBP X is very close to a random MBP Y from the stationary GD distribution, (since $d(X, Y) < 1$), we can say that X is also effectively drawn from the same GD distribution. □

Finally, I can use this result to formally define my algorithm for generating a random maximum matching (from the GD distribution) for bipartite graphs, which is to simply apply the transition to an arbitrary MBP T_{mix} times.

Here is the algorithm I provide to generate/sample an MBP for Graph G from the GD distribution, named after me (the author of this paper) <3.

Algorithm 4 amitjoshi24-algorithm

```
procedure AMITJOSHI24-ALGORITHM(G)                ▷ Takes in G as input
    apply augmenting path algorithm to generate arbitrary MBP X
    for t ∈ T_mix do
        X ← P(X)                                  ▷ Apply P to X
    end for
    return X
end procedure
```

5 Conclusion

In conclusion, this research shows a Markov Chain irreducible over all MBPs and aperiodic, and shows that the state distribution is dependent on the graph G (GD distribution), not on the initial

matching given to the algorithm. Furthermore, it shows how the stationary distribution gives positive probability to all MBPs. I show how to create a Markov Chain over all MBPs whose unique stationary distribution is uniform. I also show how to construct a coupling argument between two MBPs, both among swap transitions and augmenting path transitions (using induction) assuming some conjectures hold true. Another contribution I have made is giving an algorithm that finds a simple path (path with no repeated nodes) from s to t in such a way that any simple path from s to t is chosen with positive probability and calculates the probability with which that path is chosen.

6 Future Work

Future Research should look into determining what the stationary distribution (GD distribution) is. For $\hat{P}$, future research can also look into more efficient ways of seeing if the augmenting path transition be modified to lead to a uniform stationary distribution in the Markov Chain $M2$ (rather than the $O(|V||E|^2)$ per augmenting path method of calculating probabilities). Future research should look into either proving the conjectures described in this paper or look into other methods of bounding mixing time such as Conductance such as from (Jerrum and Sinclair) or canonical paths such as the one from (Dyer et al., 2017). Future work can also apply ideas from this paper to other problems.

7 Proofs of Lemmas

Lemma 1. If (v, w) exists in any MBP of G, and $(v, w) \notin E^i$ then either swap transition or augmenting path transition will be found such that (v, w) will appear in i', where i' is the result of applying transition P to i. The converse holds true as well.

Proof. If either v or w is unmatched in i, a swap transition can be made by unmatching the remaining node (removing one edge) so that now both v and w are unmatched, allowing for them to be matched to each other by adding (v, w) to i to give i'. One edge was removed and one edge was added so the total number of edges in i' is still m, so this is still an MBP.

If (v, w) exists in some MBP of G, then that means that the MBP of the subgraph G' must be of size $(m-1)$, so that adding (v, w) back would make m edges. Ford-Fulkerson's algorithm states that if and only if a maximum bipartite matching hasn't been reached, an augmenting path will be found. Note that $i(G')$ will initially have $m-2$ edges since $(v, f_i(v))$ and $(w, f_i(w))$ were just removed from the graph. Hence, given that (v, w) is in an MBP of G, then $i'(G')$ needs to contain $(m-1)$ edges, meaning that an augmenting path has to be found in $i(G')$ to gain one more edge. QED. □

Lemma 2. Any augmenting path in $i(G')$ must either start with $f_i(w)$ or end with $f_i(v)$. If it does both, then matches for nodes get "cycled" meaning that matched nodes stay matched and unmatched nodes stay unmatched, but the matches themselves are different.

Proof. Let's say that an augmenting path found in $i(G')$ includes neither $f_i(w)$ nor $f_i(v)$. Then, this augmenting path could also have been found in i. As every augmenting path increases the number of edges by 1, then finding an augmenting path in i achieves i' having $m+1$ edges, which is a contradiction since m is the size of any MBP of graph G.

Hence, it has to either contain $f_i(w)$ or $f_i(v)$.

Case the augmenting path contains $f_i(w)$ (node in L):

For readability, as we are talking about an augmenting path found in $i(G')$, I will denote $f_{i(G')}$ simply as f. $f_i(w)$ must occur at the start of the augmenting path since $f_i(w)$ is unmatched in $i(G')$ (since w was removed in G') and the only incoming edges in $Res_L(i(G'))$ leading into $l \in L$ are of the form $(f(l), l)$, we know that $f(f_i(w)) = -1 \implies (f(f_i(w)), f_i(w)) \notin E$ since $f_i(w)$ is unmatched in $i(G')$. Intuitively, there is no way to even reach $f_i(w)$ during the augmenting path dfs unless that is the initial unmatched node on the left passed in as a parameter. So, $f_i(w)$ must occur in the beginning of the augmenting path.

Case the augmenting path contains $f_i(v)$ (node in R):

For readability, as we are talking about an augmenting path found in $i(G')$, I will denote $f_{i(G')}$ simply as f. The only outgoing edges in $Res_R(i(G'))$ from $r \in R$ are of the form $(r, f_i(r))$ but since v is unmatched in $i(G')$ (since v was removed in G'), $f(f_i(v)) = -1 \implies (f_i(v), f(f_i(v))) \notin E$. Intuitively, once at $f_i(v)$, the augmenting path dfs can't move to any other node. So, $f_i(v)$ must occur at the end of the augmenting path.

If the augmenting path both starts with $f_i(w)$ and ends with $f_i(v)$ then this is a "cycle" then each node that is matched stays matched and each node that is unmatched stays unmatched. This is because $f_i(w)$ and $f_i(v)$ used to be matched, were unmatched in $i(G')$, and then matched to different nodes in $i'(G')$ and hence j. Every other node in the augmenting path was already matched in $i(G')$ (by the same logic as earlier in this sub-theorem, unmatched nodes on the left □

Lemma 3. The state space Ω of this Markov Chain $M1$ are MBPs, as any transition will only lead to another MBP.

Proof. For the swap transitions, one edge is removed and one edge is added, so the net gain in edges is zero, and hence a swap transition on an MBP will lead to another MBP.

For the augmenting path transitions, note that in $i(G')$, $f_i(v)$ and $f_i(w)$ are unmatched since nodes v and w were removed. If such an augmenting path is found, then let j equal $i'(G')$ with v,w,and (v, w). Formally, let $j = i' + (\{v, w\}, \{(v, w)\})$. In other words, let j be $i'(G')$ after adding back in the v,w,(v, w) that were taken out. This state will still be in Ω since first we unmatched v, unmatched w (lost 2 edges), then matched v to w (gained 1 edge) and then finally the augmenting path led to gaining 1 more edge in G', for a net gain in edges of 0. In other words, this new state is still a MBP because the size of the matching is the same.

And hence, we have transitioned from MBP i to another MBP j. □

Lemma 4. G^{ij} is a collection of disjoint paths/cycles

Proof. Given that each node in G^{ij} has degree $0, 1, 2$ (where degree is the number of edges touching the node), I will prove that G^{ij} is a collection of disjoint paths/cycles. First, note that disjointness holds simply by seeing that if a path/cycle wasn't disjoint from another path/cycle, then both of them would be considered to be of the same path/cycle, and hence disjoint from every other path/cycle in G^{ij}. Next, to prove that every set of nodes is indeed part of a path/cycle (or it is an "island" unmatched in both i and j), we can try to find such paths/cycles. To find paths (specifically, simple paths), find a node with degree 1, this will be an endpoint of a simple path. Keep traversing this path (without visiting visited nodes, as this path is simple) using either breadth-first-search

(BFS) or depth-first search (DFS) until another node is reached with degree 1. This will be the other endpoint of this simple path. It is guaranteed that such a node exists. This is because every node along this path except for the endpoints (middle nodes) has degree 2 (because if it had degree 0, this node would never be reached). Upon the BFS/DFS reaching each of these middle nodes, a middle node has only 1 unvisited child, as a max degree of 2 means that one of those edges was used to get to the middle node, leaving only 1 more unused edge. Eventually a middle node has to lead to an endpoint (node with degree 1) since otherwise, there would be an infinite amount of nodes in a finite graph.

Algorithm 5 find-simple-paths

```
1:  procedure FIND-SIMPLE-PATHS(G^ij)                    ▷ Parameters: Graph G^ij
2:      visited ← ∅
3:      endpoints ← all degree 1 nodes in G^ij
4:      paths ← []
5:      while endpoints ≠ ∅ do
6:          start ← any endpoint
7:          queue ← [start]
8:          path ← []
9:          while len(queue) > 0 do
10:             cur ← queue.poll()
11:             add cur to visited
12:             add cur to path
13:             for neighbor nb of cur do            ▷ cur has a maximum of two neighbors
14:                 if nb ∉ visited then
15:                     add nb to queue
16:                 end if
17:             end for
18:         end while
19:         add path to paths
20:     end while
21: end procedure
```

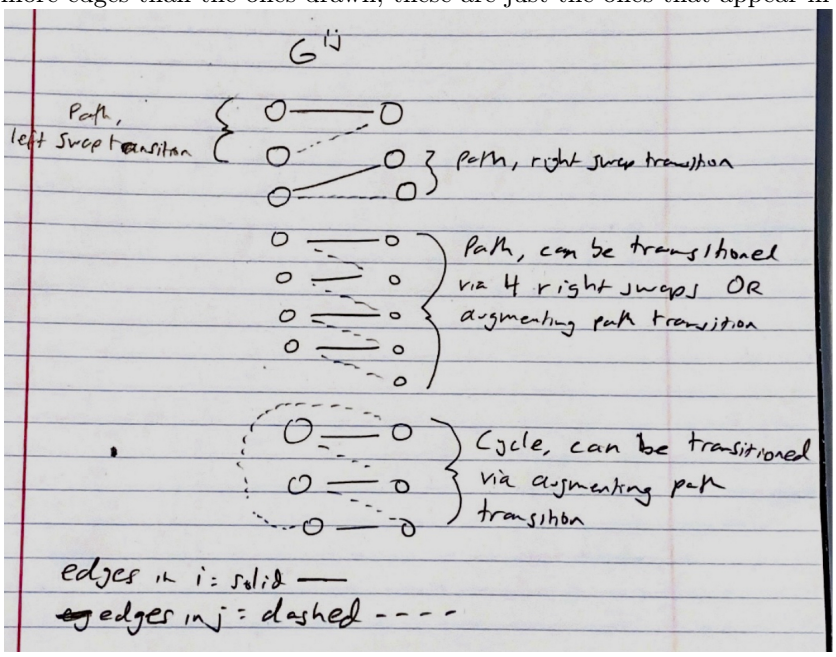
Note that they are all disjoint. To find the cycles, simply start at any remaining node (after all paths have been found, so they all have degree 2), and use BFS/DFS like before until it arrives back at the first node you picked (guaranteed to happen since every remaining node now has degree 2, and there are only finitely many nodes). Note that the BFS queue has length at most 1, and if in any iteration a neighbor does not get added, then the queue will be empty, and the middle loop will terminate.

Algorithm 6 find-simple-cycles

```
1:  procedure FIND-SIMPLE-CYCLES(G^ij)                   ▷ Parameters: Graph G^ij
2:      visited ← ∅
3:      nodes ← all remaining nodes in G^ij
```

```
    cycles ← []
    while endpoints ≠ ∅ do
        start ← any node ∈ nodes
        queue ← [start]
        cycle ← []
        while len(queue) > 0 do
            cur ← queue.poll()
            add cur to visited
            add cur to cycle
            for neighbor nb of cur do                ▷ cur has a maximum of two neighbors
                if nb ∉ visited then
                    add nb to queue
                else if nb = start then
                    add nb to cycle                  ▷ queue will be empty
                end if
            end for
        end while
        add cycle to cycles
    end while
end procedure
```

To help the reader visualize G^{ij}, here is a diagram of what it can look like. Note that there can be more edges than the ones drawn, these are just the ones that appear in MBPs i and j.



□

Lemma 5. In any of these disjoint paths/cycles, the number of edges from each matching must be the same.

Proof. Next, I will show that in any of these disjoint paths/cycles, the number of edges from each matching must be the same. Proof by contradiction: Let's say that a particular path/cycle $A = (V^A, E^A)$ has more edges from E^i compared to E^j. Then, it would be true that if j were to remove the edges from $E^A \cap E^j$ and add the edges from $E^A \cap E^i$ then j would have more edges than before, since A is disjoint from all other cycles/paths that were formed. This would contradict the fact that j is a MBP. Without loss of generality, A can not have more edges from E^j compared to E^i either. And hence $E^A \cap E^i = E^A \cap E^j$. □

Lemma 6. In a disjoint path, which has exactly two nodes with degree 1 (endpoints) in G^{ij}. Either both endpoints are in L or both are in R.

Proof. Proof by contradiction. Assume there exists a node of degree 1 on the left and another node of degree 1 on the right. Hence, this path can be thought of as starting on the left and ending on the right. Note that each edge switches sides (one endpoint is in L and the other endpoint is in R). Then, the path can be thought of as $l_1, r_1, l_2, r_2, \ldots, r_n$. But that would mean that there are an odd number of edges in the path, and hence E^i and E^j did not contribute the same number of edges to the path, which contradicts what we proved earlier in Lemma 5. Hence, the path can be thought of as either $l_1, r_1, l_2, r_2, \ldots l_n$ (left-path) or as $r_1, l_1, r_2, l_2, \ldots r_n$ (right-path). □

Lemma 7. One of the two endpoints of a disjoint path is touching an edge in E^i and the other is touching an edge in E^j.

Proof. Each disjoint paths has two endpoints (nodes with degree 1). I claim that one of these two endpoints is touching an edge in E^i and the other is touching an edge in E^j (and these are the only edges they are touching, since both these nodes have degree 1). Consider left-paths first, with two endpoints in L first. Since this path can be thought of as starting on the left and also ending on the left (by Lemma 6), and this path jumps sides with each edge taken so the nodes along this path can be thought of as $l_1, r_1, l_2, r_2, \ldots, l_n$. Realize that l_1 and l_n are the two nodes with degree 1 in this path. We know that each node can only touch at most 1 edge from E^i and at most 1 edge from E^j, so the matching the edges along this path were taken from must alternate. Specifically, (l_i, r_i) must be from a different matching than (r_i, l_{i+1}) otherwise r_i would be matched to two nodes in the same matching. Likewise, (l_{i+1}, r_{i+1}) must have been taken from a different matching than (r_i, l_{i+1}) otherwise l_{i+1} would be matched to two nodes in the same matching. Hence, the matchings each edge was taken from must also alternate. We can say that the edges of the form (l_i, r_i) were all taken from the first matching and say that the edges of the form (r_i, l_{i+1}) were all taken from the second matching. Hence, we know that (l_1, r_1) had to have been taken from a different matching than (r_{n-1}, l_n).

Without loss of generality, the same result holds for right-paths of the form $r_1, l_1, r_2, l_2, \ldots r_n$. □

Lemma 8. Any of these cycles has $2n$ nodes for some n, where n of those are in L and n are in R.

Proof. Direct proof:

Let's say that the BFS from above comes up with the cycle $v_1, v_2, \ldots, v_k, v_1$. Note that in this cycle, each node has degree 2 since it is connected to two other nodes in G^{ij} by construction. Simply counting the edges in the cycle shows that there are k edges.

Since each edge in E^{ij} crosses sides, and v_1 is on the same side as v_1 (because they are the same vertex) there must be an even number of edges in this cycle. More generally, if the parity of number of edges taken on the cycle since v_1 to some v_i is odd, then v_i is on the opposite side, if it is even, then v_i is on the same side.

Proof by induction:

Base case: 0 edges taken since v_1, so $v_i = v_1$ and is hence on the same side as v_1.

Base case: 1 edge taken since v_1, so $v_i = v_2$. As $(v_1, v_2) \in E^{ij}$ and each edge in E^{ij} has one endpoint in L and the other in R. Hence v_2 is on the opposite side of v_1.

Inductive hypothesis (IH): assume property holds true for $m-1$ edges taken

Inductive case: assuming IH, prove that property holds for m edges taken since v_1 on the eulerian circuit.

Case $m-1$ is even, then v_m is on the same side as v_1 by IH. The next edge on the eulerian circuit is (v_m, v_{m+1}). Now, m (odd) edges have been taken, and v_{m+1} is on the opposite side of v_m since $(v_m, v_{m+1}) \in E^{ij}$ in which one endpoint must be in L, and the other must be in R. Hence, v_{m+1} is on the opposite side of v_1.

Case $m-1$ is odd, then v_m is on the opposite side as v_1 by IH. The next edge on the eulerian circuit is (v_m, v_{m+1}). Now, m (even) edges have been taken, and v_{m+1} is on the opposite side of v_m since $(v_m, v_{m+1}) \in E^{ij}$ in which one endpoint must be in L, and the other must be in R. Hence, v_{m+1} is on the same side of v_1.

Note that since $(v_k, v_1) \in E^{ij}$ then v_1 and v_k lie on opposite sides of the bipartite graph. Hence, there had to have been an odd number of edges taken from v_1 to v_k. But, there is one more edge in the eulerian circuit, namely (v_k, v_1). Hence, the total number of edges (k) in the eulerian circuit, which is the same as the cycle itself, is even. Express $k = 2n$ since k is even.

Observe that $v_1, v_3, v_5, \ldots v_{k-1}$ lie on the same side and $v_2, v_4, v_6, \ldots v_k$ lie on the same side by the induction proved earlier.

Counting the terms on the first side yields:

$$\frac{(k-1)-1}{2} + 1 \tag{37}$$

$$= \frac{2n-2}{2} + 1 \qquad k = 2n \tag{38}$$

$$= n - 1 + 1 \tag{39}$$

$$= n \tag{40}$$

Counting the terms on the second side yields:

$$\frac{k-2}{2} + 1 \tag{41}$$

$$= \frac{2n-2}{2} + 1 \qquad k = 2n \tag{42}$$

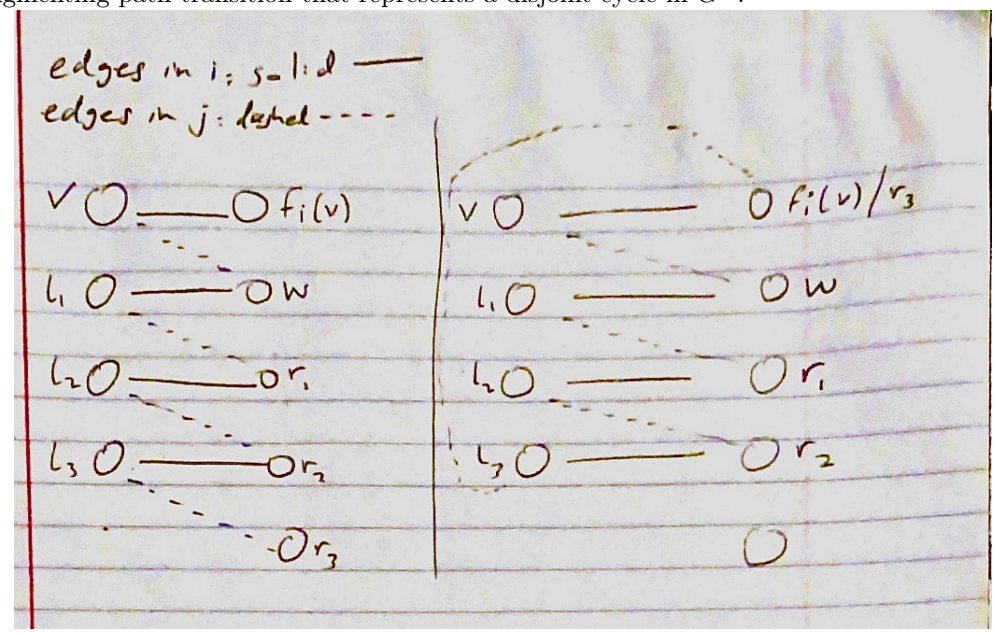
$$= n - 1 + 1 \tag{43}$$

$$= n \tag{44}$$

Hence, in this cycle, there are n nodes in L and n nodes in R, for some n. QED. □

Lemma 9. For every augmenting path transition from MBP i to MBP j there exists a unique opposite augmenting path transition from j to i.

Proof. WLOG, let the augmenting path transition that takes i to j is selecting $(l, f_j(l)) = (v, w)$ in step 1. Then, augmenting path $l_1, r_1, l_2, r_2, \ldots l_n, r_n$ is taken. By Lemma 2, we know that either $l_1 = f_i(w)$ or $r_n = f_i(v)$. WLOG, let's say $l_1 = f_i(w)$. Then, the augmenting path transition that takes j to i is selecting $(l_n, r_{n-1}) = (v, w)$ in step 1. Then, augmenting path $l_{n-1}, r_{n-2}, l_{n-2}, r_{n-3}, \ldots, l_1, f_j(l), l, f_i(l)$. Here are two diagrams that illustrates this, with solid edges being from matching i and dashed edges being from matching j. The left one is of an augmenting path transition that represents a disjoint path in G^{ij}, and the right one is of an augmenting path transition that represents a disjoint cycle in G^{ij}.



If this transition from i to j represents a disjoint path in G^{ij}, then this augmenting path transition is unique, as this disjoint path is a tree, and the path between any two nodes of a tree is unique. If this transition from i to j represents a disjoint cycle in G^{ij}, then this augmenting path transition is one of k transitions from j to i, where $2k$ edges are in this disjoint cycle, where k are in i and k are in j. Hence, picking any of these k edges in j as (v, w) in step 1 can lead to an augmenting path transition from i to j (which is unique, as G' will remove v and w, turning this cycle into a path), and similarly picking any of these k edges in i as (v, w) in step 1 can lead to an augmenting path transition from j to i. □

Lemma 10. $d(X, Y) = d(Y, X)$

Proof. Let $e_{x_1}, ..., e_{x_d}$ be edges in E^X but not in E^Y. Then, we know that X and Y have exactly $(m - d)$ edges in common, since X and Y are both MBPs, we know that they each have m edges. Of which, Y shares $(m - d)$ in common with X, leaving d edges in E^Y but not in E^X. Hence, $d(Y, X) = d(X, Y)$. □

Lemma 11. The distance function d obeys the triangle inequality. As in $d(A, C) \leq d(A, B) + d(B, C)$ for MBPs A, B, C.

Proof. Let $k_1 = d(A,B), k_2 = d(B,C)$. Then, one could remove the k_1 edges in A but not in B, add the k_1 edges in B and not in A. Then, one could remove the k_2 edges in B but not in C, add the k_2 edges in C but not in B. In total, a maximum of $k_1 + k_2$ edges could have been removed/added.

Alternatively, consider $\Delta(A,B)$ as the set of all edges in A but not in B, or vice versa. Note how $|\Delta(A,B)| = 2d(A,B)$, since for every edge in A but not in B, there must exist another edge in B but not in A. Likewise consider $\Delta(B,C)$. It is sufficient to prove the following claim, that $\Delta(A,C)$ is contained in $\Delta(A,B) \cup \Delta(B,C)$:

$$\Delta(A,C) \subseteq \Delta(A,B) \cup \Delta(B,C) \tag{45}$$

Proof of claim:

Rewrite the claim the following way:

$$\forall e, \text{if } e \in \Delta(A,C) \implies e \in \Delta(A,B) \cup \Delta(B,C) \tag{46}$$

Proof by contraposition. Let $e \notin \Delta(A,B) \cup \Delta(B,C)$. So, $e \notin \Delta(B,C)$ and $e \notin \Delta(A,B)$. Given that $e \notin \Delta(B,C)$, we have two cases:

In one case, $e \notin E^B$ and $e \notin E^C$. Since $e \notin \Delta(A,B)$ we know that $e \notin E^A$. Since $e \notin E^A \wedge e \notin E^C \implies e \notin \Delta(A,C)$.

In the other case, $e \in E^B$ and $e \in E^C$. Since $e \notin \Delta(A,B)$, we know that $e \in E^A$. Hence, we have that $e \in E^A \wedge e \in E^C \implies e \notin \Delta(A,C)$.

So, $\Delta(A,C) \subseteq \Delta(A,B) \cup \Delta(B,C)$. QED.

$\lvert\Delta(A,C)\rvert \leq \lvert\Delta(A,B) \cup \Delta(B,C)\rvert$	Follows from claim just proved	(47)
$\lvert\Delta(A,C)\rvert \leq \lvert\Delta(A,B)\rvert + \lvert\Delta(B,C)\rvert - \lvert\Delta(A,B) \cap \Delta(B,C)\rvert$	Cardinality of set union	(48)
$\lvert\Delta(A,C)\rvert \leq \lvert\Delta(A,B)\rvert + \lvert\Delta(B,C)\rvert$	Cardinality is always positive	(49)
$2d(A,C) \leq 2d(A,B) + 2d(B,C)$		(50)
$d(A,C) \leq d(A,B) + d(B,C)$	divide by 2	(51)

□

8 Acknowledgements

I express my thanks to my advisor Dr. Shuchi Chawla. Lastly, I have decided to make my paper accessible at: https://github.com/amitjoshi24/randomizing-maximum-bipartite-matchings

9 References

1 A.B. Kahn, Topological sorting of large networks, Communications of the ACM, 5 (1962) N.11, 558–562

2 Ali Ibrahim. Maximum Independent Set in Bipartite Graphs *Ali Ibrahim Site*, 2020.

3 Anant Jindal, Gazar Kochar, and Manjish Pal. Maximum Matchings via Glauber Dynamics In *arXiv preprint arXiv:1107.2482*, 2011.

4 Martin Dyer, Mark Jerrum, and Haiko Muller. On the switch Markov chain for perfect matchings In *arXiv preprint arXiv:1501.07725 Journal of the ACM, 64:Article 2, 2017.*

5 Kleinberg, J. and Tardos, E., 2014. Algorithm design. Boston: Pearson/Addison-Wesley.

6 Mitzenmacher, M. and Upfal, E., 2005. Probability and Computation. 1st ed. Cambridge, UK: Cambridge University Press.

7 M. Jerrum and A. Sinclair. Approximating the permanent. SIAM Journal on Computing, 18(6):1149–1178, 1989.

8 A. Z. BRODER, How hard is it to marry at random? (On the approximation of the permanent), in Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 50-58; Erratum in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, p. 551, Association for Computing Machinery, New York.

9 Miklos, Istvan Krész, Miklós. (2020). Counting Maximum Matchings in Planar Graphs Is Hard.

10 A. Lyons, "Polynomial-Time Approximation of the Permanent," Course project for MATH, vol. 100, 2011.

11 P. Kasteleyn. Graph theory and crystal physics. Graph Theory and Theoretical Physics, pp. 43–110. Academic Press, 1967. 2

10 Appendix

For convenience, I provide a screenshot and link [1] to an amazing site for learning about Maximum Matching and the Augmenting Path algorithm.

[1] https://ali-ibrahim137.github.io/competitive/programming/2020/01/02/maximum-independent-set-in-bipartite-graphs.html

Below you can find the code to solve the **MCBM** problem using augmenting path algorithm.

```
vector<int>graph[MX];
bool vis[MX];
int match[MX];
bool dfs(int node){
        if(vis[node])return 0;
        vis[node] = 1;
        for(auto nx:graph[node]){
                if(match[nx]==-1 || dfs(match[nx])){
                        match[node] = nx;
                        match[nx] = node;
                        return 1;
                }
        }
        return 0;
}
// inside main()
memset(match, -1, sizeof match);
while(1){
        memset(vis, 0, sizeof vis);
        bool cont = 0;
        for(int i=1;i<=n;i++){
                if(match[i]==-1)cont|=dfs(i);
        }
        if(cont==0)break;
}
int MCBM = 0;
for(int i=1;i<=n;i++){
        if(match[i]!=-1)MCBM++;
}
```