Randomizing Maximum Matchings In Lopsided Bipartite Graphs Using Markov Chain Monte Carlo Coupling

Amit Joshi
Email: amitjoshi24@utexas.edu

December 2021

1 Abstract

In this paper I explore randomizing maximum bipartite matchings (MBPs) using Markov Chain Monte Carlo Coupling. I show that given certain constraints on the degree of the vertices (lopsidedness) in the bipartite graph, and the size of the MBP, it is possible to generate a uniformly random MBP. I give an algorithm to randomize an arbitrary MBP assuming those certain constraints on the graph, and give bounds on runtime for my algorithm.

2 Introduction

2.1 Related Work

Prior work has been done on generating a random maximum bipartite matching with high probability (Jindal et al., 2011). 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.

However, there hasn't been any work done on taking an existing MBP and subsequently randomizing it. In this paper I explore an algorithm that will take an existing MBP (which can be computed via a variety of known algorithms) and perform a series of operations that will take it asymptotically closer to a random MBP.

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 $(l, r), l \in L, r \in R$.

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 $\forall r_1, r_2 \in R, r_1 \neq r_2$, $(r_1, r_2) \notin E$ (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.

Definition 2.1 (Lopsided)**.** Lopsided is a term I use to mean bipartite graphs in which one of the sets, L, only contains vertices that have a degree at least k and the other set, R, only contains vertices that have degree at most Δ, where $k > 4\Delta$. Also, a lopsided bipartite graph must have a maximum matching of size L. Note that degree of a vertex is the number of edges in E touching it. Finally, a Lopsided graph is one in which for every MBP, each $l \in L$ has a neighbor in $r \in R$ that is unmatched.

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 (Jindal et al., 2011).

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.

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 this arbitrary matching and randomize it, provided the bipartite graph is indeed lopsided.

2.3 Markov Chain Monte Carlo

Markov Chain Monte Carolo (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 (Jindal et al., 2020). 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:

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

A distribution π is called stationary if $P\pi = \pi$, meaning that the transition matrix does not affect the distribution.

A Markov Chain has a *unique* stationary distribution (is ergodic) if the following conditions hold (Jindal et al., 2011):

- Irreducible: $\forall i, j \in \Omega$, $\exists t$ such that $P_{i,j}^t > 0$
- Aperiodic: $\forall i \in \Omega$, $gcd(t, P_{i,i}^t) = 1$

2.3.2 Coupling

Coupling is a technique to estimate the expected difference between a state and a uniformly random state in a Markov Chain (Jindal et al., 2020). Consider the tuple of stochastic processes $(X, Y) \in \Omega \times \Omega$. I attempt to perform the same transition to both X and Y, so note that:

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

Define a distance function $d(a, b)$ which computes the difference between two states in Ω. The aim of coupling is to decrease the *expected* distance between X and Y.

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 is small (< 1), I consider X to be uniformly random.

3 Methods

Define the lopsided (and connected) 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 $(l, r), l \in L, r \in R$. Given the graph is lopsided, there exists a maximum matching of size $|L|$, which can be found via the Augmenting Path algorithm or by a reduction to max-flow and using a max-flow algorithm. Everything below concerns itself solely with bipartite graphs that meet the lopsided requirement.

Theorem 1. *Given G is lopsided, $|R| > |L|$.*

Proof. Given G lopsided, we know each vertex in L has degree at least k, and each vertex in R has degree at most Δ. Assume by way of contradiction that $|R| \leq |L|$. Then, in expectation, a vertex in R degree k, meaning that $\exists r \in R$ with degree at least k. But, as we know $k > 4\Delta$, which contradicts the fact that G is lopsided, as r can only have degree at most Δ. □

Define a function $\delta : V \to \mathbb{Z}$ where $\delta(v)$ equals the degree of v.

Now, let's define a function $\tau : V \to (0, 1]$ which represents the probability that $v \in V$ is unmatched (taken uniformly under all MBPs of G). Note that $\forall l \in L, \tau(l) = 0$ since $\forall l \in L$, l has a matching. Below I show how to calculate $\tau(r)$ for all $r \in R$.

Theorem 2. *Given size of the MBP is $|L|$, it is possible to calculate the probability that a particular vertex in R is matched in G. Not only that, it is efficiently computable. And hence τ is defined for all vertices in V.*

Proof. Since the size of the MBP is $|L|$ we know that each $l \in L$ is matched with a vertex. Now, we consider the probability that $r \in R$ is not matched with any vertex in L (taken uniformly under all MBPs of G).

Consider edge (l, r), where $l \in L$ and $r \in R$. Note that since l is indeed matched, the probability that (l, r) is taken into the MBP is $\frac{1}{\delta(l)}$ because exactly one of l's $\delta(l)$ edges are taken. It follows that the probability that (l, r) is not taken into the MBP is $1 - \frac{1}{\delta(l)}$.

Notice that for r to be unmatched, all edges containing r (denoted as (l_i, r)) must be NOT taken into the MBP. Hence, mathematically, the probability that r is unmatched is:

$$\Pi_{i=1}^{\delta(r)} \left(1 - \frac{1}{\delta(l_i)}\right) = \tau(r)$$

□

Now, let's define the Markov Chain and its transition matrix P. Let the finite state space Ω be the set of all possible maximum matchings for G. For any state $i \in \Omega$, let the transition work as follows:

1. Select vertex $v \in L$ uniformly at random
2. Select vertex $w \in R$ random from all such w such that $(v, w) \in G$, such that w is selected with probability $\frac{1}{\tau(w)*\delta(v)}$
3. If w is unmatched, match v to w

Intuitively, I wanted the probability that w is matched with (after v is selected) to be independent of $\delta(w)$. I wanted the probability that w is matched with to be uniformly random across all neighbors of v. Note that once v is selected, it matches with each of its neighbors with $\frac{1}{\delta(v)}$ probability. Below is a mathematical explanation of why this construction works.

$$\begin{aligned} Pr[w \text{ matched}] &= Pr[w \text{ initially unmatched}] * Pr[w \text{ chosen in step 2}] \\ &= \tau(w) * \frac{1}{\tau(w) * \delta(v)} \\ &= \frac{1}{\delta(v)} \end{aligned}$$

Theorem 3. *The above Markov Chain is irreducible and aperiodic, and hence is ergodic has a unique stationary distribution.*

Proof. First, let me show this Markov Chain is irreducible. For it to be irreducible, given any two states $i, j \in \Omega$, it must be possible to reach j starting from i with nonzero probability. I give an algorithm to reach j from i.

Define f_i to be a mapping from a vertex to its matching vertex (or to -1 in the case that the vertex is unmatched), similarly define f_j.

This can be accomplished via an algorithm extremely similar to Kahn's topological sort (Kahn 1962). In this algorithm, tasks are processed in a certain order, and once a task is processed, it frees up other tasks that depended on it.

Kahn's topological sort works as is in the case that there are no circular dependencies (e.g., l_1 needs to match with r_1 but is currently matched to r_2, and l_2 needs to match with r_1 but is currently matched to r_1). We can think of there being $|L|$ tasks, and that each task is assigning a vertex l to $f_j(l)$. If $f_j(l)$ is currently matched with another node, say l' ($l' = f_i(f_j(l))$, then we consider l to be a dependency of l', as l' must first be processed (and matched with another node) before l can be matched with $f_j(l)$. Note that by construction, each task is dependent on at most one other task. Tasks with zero dependencies can be processed immediately and are put in the set $zeroIn$.

Let's see how Kahn's works in the case that there are no circular dependencies.

Algorithm 1 uses Kahn's topological sort to use the proposed Markov Chain and transition to go from state i to state j in the case of no circular dependencies.

Algorithm 1 iToj

procedure iToj(i, j)
  $zeroIn \leftarrow$ copy of L
  $dependencyMap \leftarrow \emptyset$
  $reverseDependencyMap \leftarrow \emptyset$
  for $l \in L$ **do** ▷ add dependencies
    $target \leftarrow f_j(l)$
    $targetCurrentMatch \leftarrow f_i(target)$
    if $targetCurrentMatch \neq -1$ **then**
      $dependencyMap[l] \leftarrow targetCurrentMatch$ ▷ l depends on $targetCurrentMatch$
      $reverseDependencyMap[targetCurrentMatch] \leftarrow l$
      remove l from $zeroIn$
    end if
  end for
  $numProcessed \leftarrow 0$
  while $numProcessed < |L|$ **do**
    $cur \leftarrow$ any element from $zeroIn$
    set $f_i(f_i(cur))$ to -1 ▷ unmatch on right side
    set $f_i(cur)$ to $f_j(cur)$
    set $f_i(f_i(cur))$ to cur
    $numProcessed \leftarrow numProcessed + 1$
    $dependsOnCur \leftarrow reverseDependencyMap[cur]$
    add $dependsOnCur$ to $zeroIn$
    remove cur from $reverseDependencyMap$
    remove $dependsOnCur$ from $dependencyMap$
  end while
  ▷ at this point, $i = j$
end procedure

The modification that I propose is that in the case that when there is a circular dependency, take any vertex v in the dependency and match it to one of its unmatched neighbors in R, (call this node $temp$ now) which is guaranteed to exist since G is lopsided. This removes the circular dependency and Kahn's goes through since there is at least 1 vertex that can be processed in $zeroIn$. Whenever possible, assign v to $f_j(v)$ (when the circle finishes processing). Note that it is guaranteed that v will no longer be part of a circular dependency as no other vertex needs to be matched to $f_j(v)$. Note how then it is guaranteed that there will only be at most 1 $temp$ at a time, because at most 1 circle will be processing. Once that circle is done processing, $temp$ will be assigned to its final matching ($f_j(temp)$).

Let's see the modified algorithm now:

Algorithm 2 uses a modified version Kahn's topological sort to use the proposed Markov Chain and transition to go from state i to state j even with the case of circular dependencies.

Algorithm 2 iTojModified

procedure iTojModified(i, j)
  $zeroIn \leftarrow$ copy of L
  $dependencyMap \leftarrow \emptyset$
  $reverseDependencyMap \leftarrow \emptyset$
  for $l \in L$ **do** ▷ add dependencies
    $target \leftarrow f_j(l)$
    $targetCurrentMatch \leftarrow f_i(target)$
    if $targetCurrentMatch \neq -1$ **then**
      $dependencyMap[l] \leftarrow targetCurrentMatch$ ▷ l depends on $targetCurrentMatch$
      $reverseDependencyMap[targetCurrentMatch] \leftarrow l$
      remove l from $zeroIn$
    end if
  end for
  $numProcessed \leftarrow 0, temp \leftarrow -1$
  while $numProcessed < |L|$ **do**
    if $|zeroIn| = 0$ **then**
      $temp \leftarrow$ any unprocessed task
      $unmatchedTempNeighbor \leftarrow$ any unmatched neighbor of $temp$
      set $f_i(temp)$ to $unmatchedTempNeighbor$ then set $f_i(f_i(temp))$ to $temp$
    end if
    $cur \leftarrow$ any element from $zeroIn$
    set $f_i(f_i(cur))$ to -1 ▷ unmatch on right side
    set $f_i(cur)$ to $f_j(cur)$ then set $f_i(f_i(cur))$ to cur
    $numProcessed \leftarrow numProcessed + 1$
    $dependsOnCur \leftarrow reverseDependencyMap[cur]$
    add $dependsOnCur$ to $zeroIn$
    remove cur from $reverseDependencyMap$
    ▷ check if circle has finished processing and $temp$ can be processed
    if $f_i(f_j(temp)) = -1$ **then** ▷ $temp$ can now be matched with $f_j(temp)$
      set $f_i(f_i(temp))$ to -1 ▷ unmatch on right side
      set $f_i(temp)$ to $f_j(temp)$ then set $f_i(f_i(temp))$ to $temp$ then set $temp \leftarrow -1$
    end if
  end while ▷ at this point, $i = j$
end procedure

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. Note that in the transition, if $v \in L$ is randomly selected in step 1, and v is already matched with $w \in L$, w is selected in step 2 with positive probability $\frac{1}{\tau(w)*\delta(v)}$, and the MBP will stay the same.

Hence, by the Fundamental Theorem of Markov Chains, this Markov Chain is ergodic and has a unique stationary distribution.

□

Theorem 4. *The uniformly random distribution is stationary for the above Markov Chain.*

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, $Pr[i] = \frac{1}{|\Omega|}$.

Let j be accesible via one transition. This means that for some $v \in L$, and some $w \in R$, v was matched to w, where $w = f_j(v)$. As we saw earlier, w was matched with probability $\frac{1}{\delta(v)}$. Note that v was originally matched to $f_i(v)$.

Since j also comes from the uniformly random distribution, $Pr[j] = \frac{1}{|\Omega|}$. Note that j can transition to i by selecting v and matching it to $f_i(v)$. As we saw earlier, this occurs with probability $\frac{1}{\delta(v)}$.

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 P. Hence, the uniformly random distribution is stationary.

□

Theorem 5. *The above Markov Chain will eventually converge to a uniformly random state.*

Proof. By Theorem 3, there is only one stationary distribution, and by Theorem 4, it is the uniformly random distribution, so eventually, the Markov Chain will arrive at a uniformly random state. □

Let's now define the coupling (X, Y). X is an arbitrary maximum bipartite matching of G. Y is a uniformly 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 uniformly random state, as Theorem 4 showed that the uniformly random distribution is stationary.

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 $f_X(u) = f_X(v)$ if $(u, v) \in X$. Note that $f_X(u) = -1$ if u is unmatched. Similarly, define f_Y.

Let's define a distance function for this coupling d_L (distance from the left). Define $d_L(X, Y)$ to be the number of vertices $l \in L$ such that $f_X(l) \neq f_Y(l)$. We consider X to be uniformly random when $d_L(X, Y) < 1$. Also define a distance function $d_R(X, Y)$ to be the number of vertices $r \in R$ such that $f_X(r) \neq f_Y(r)$.

Theorem 6. $d_R(X, Y) = 2 * d_L(X, Y)$

Proof. Consider all $l \in L$ such that $f_X(l) \neq f_Y(l)$. Let $f_X(l) = r_1$ and $f_Y(l) = r_2$. Note that $f_X(r_1) \neq f_Y(r_1)$ and $f_X(r_2) \neq f_Y(r_2)$. □

Theorem 7. *The mixing time (T_{mix}) of this coupling is:* $2k|L|ln(|L|)$

Proof. To see why this is, consider the expected change in $d_L(X, Y)$ after a transition.

Let's calculate the probability that $d_L(X, Y)$ will decrease. This will happen if in the transition, a $v \in L$ is chosen such that $f_X(v) \neq f_Y(v)$ and a $w \in R$ is chosen such that w is unmatched in both X and Y. In this case, $d_L(X, Y)$ will decrease by exactly 1, as $f_X(v)$ will now equal $f_Y(v)$ (which both equal w).

Intuitively, the below equation is saying that there are $d_L(X,Y)$ choices of v such that $f_X(v) \neq f_Y(v)$ (by construction of $d_L(X,Y)$ and each is taken with probability $\frac{1}{|L|}$.

$$Pr[f_X(v) \neq f_Y(v)] = \frac{d_L(X,Y)}{|L|} \tag{1}$$

For each $w \in R$, the probability that w is matched in X can be upper bounded the following way, because each of w's incoming edges has at most $\frac{1}{k}$ probability of being taken to MBP X, and there are at most Δ incoming edges by Lopsidedness.

$$Pr[w \text{ matched in } X] \leq \frac{\Delta}{k} \tag{2}$$

$$Pr[w \text{ matched in } Y] \leq \frac{\Delta}{k} \tag{3}$$

$$Pr[w \text{ matched in } X \text{ or } Y] \leq \frac{2\Delta}{k} \qquad \text{By union bound} \tag{4}$$

$$Pr[w \text{ unmatched in both}] \geq 1 - \frac{2\Delta}{k} \tag{5}$$

Combining (1) and (5) we get that:

$$Pr[d_L(X,Y) \text{ decreases}] \geq \frac{d_L(X,Y)}{|L|} * \left(1 - \frac{2\Delta}{k}\right) = \frac{d_L(X,Y)}{|L|} * \left(\frac{k-2\Delta}{k}\right) \tag{6}$$

Now, let's calculate the probability that $d_L(X,Y)$ increases after applying a transition. Note that only happens if the $w \in R$ selected in the transition is unmatched in X, but not Y, or vice versa.

$$\text{Number of } w \text{ free in } X, \text{ but not } Y, \text{ or vice versa } \leq d_R(X,Y) \tag{7}$$

$$= 2d_L(X,Y) \tag{8}$$

$$\text{Number of } v \in L \text{ that are neighbors of a particular } w \in R \leq \Delta \tag{9}$$

$$\text{Number of } v \in L \text{ that are neighbors of any } w \text{ free in one but not the other} \leq 2d_L(X,Y)\Delta \tag{10}$$

$$Pr[\text{Picking such } v] \leq \frac{2d_L(X,Y)\Delta}{|L|} \tag{11}$$

$$Pr[d_L(X,Y) \text{ increases}] = Pr[\text{particular } w \text{ is selected given } v] \leq \frac{2d_L(X,Y)\Delta}{|L|k} \tag{12}$$

Note that it does not matter the probability with which $d_L(X,Y)$ stays the same, as we are only interested in the expected *change* in $d_L(X,Y)$.

Now, we can calculate the expected value of $d_L(X,Y)$ after applying a transition. Note that I refer to $d_L(X,Y)$ as simply d in this calculation.

$$\mathbb{E}[d_L(X^{t+1}, Y^{t+1})] \leq 1 * Pr[d \text{ increases}] - 1 * Pr[d \text{ decreases}] \tag{13}$$

$$= \frac{2\Delta d}{kn} - \left[\frac{d}{|L|} * \left(\frac{k-2\Delta}{k}\right)\right] \tag{14}$$

$$= \frac{-d(k-4\Delta)}{k|L|} \leq \frac{-d}{k|L|} \qquad \text{Since by Lopsidedness } k > 4\Delta \tag{15}$$

Hence, the expected *change* in $d_L(X, Y)$ is at most $\frac{-1}{k|L|}$ (meaning that the expected change is either more negative that that or the same).

$$\mathbb{E}[d_L(X^{t+1}, Y^{t+1})] \leq \mathbb{E}[d_L(X^t, Y^t)] \left(1 - \frac{1}{k|L|}\right) \tag{16}$$

$$\mathbb{E}[d_L(X^t, Y^t)] \leq \mathbb{E}[d_L(X^0, Y^0)] \left(1 - \frac{1}{k|L|}\right)^t \tag{17}$$

$$\leq |L| \left(1 - \frac{1}{k|L|}\right)^t \quad \text{In the case that every vertex is matched differently} \tag{18}$$

From equation (18), setting $t = 2k|L|ln(|L|)$ gives us:

$$\mathbb{E}[d_L(X^t, Y^t)] \leq |L| \left(1 - \frac{1}{k|L|}\right)^{2k|L|ln(|L|)} \tag{19}$$

$$\leq |L| \left(\frac{1}{e}\right)^{2ln(|L|)} \tag{20}$$

$$\leq |L| \left(\frac{1}{|L|^2}\right) \tag{21}$$

$$= \frac{1}{|L|} < 1 \tag{22}$$

Since after this many iterations of the transition, an arbitrary MBP X is very close to a uniformly random MBP Y, (since $d_L(X, Y) < 1$), we can say that X is also uniformly random, and the main claim of this paper is proved. QED. < 3 (that is a heart) □

Finally, I can use this result to formally define my algorithm for generating a uniformly random maximum matching for lopsided bipartite graphs, which is to simply apply the transition to an arbitrary MBP $2k|L|ln(|L|) = T_{mix}$ times.

Algorithm 3 is the algorithm I provide to generate a random maximum matching for Lopsided graph, named after me (the author of this paper).

Algorithm 3 amitjoshi24-algorithm

```
procedure AMITJOSHI24-ALGORITHM(G)        ▷ Takes in Lopsided graph G as input
    apply augmenting path algorithm to generate arbitrary MBP X
    calculate transition matrix P using function τ defined in Theorem 2
    for t ∈ T_mix do
        X ← P(X)                          ▷ Apply P to X
    end for
    return X
end procedure
```

4 Future Work

Future work should look into modifying the algorithm and/or analysis so the lopsidedness constraint isn't so constraining. For example, it should explore if it is possible to randomize a MBP even if k is not larger than 4Δ. Future work can also explore algorithms that randomize a MBP even if the size of the MBP is not $|L|$.

Another idea could be to refine the constraints for Lopsidedness so that the existence of an unmatched r for each $l \in L$ in every MBP would either be implied by the other constraints or the proof for irreducibility works without this constraint. Note that even if that constraint of Lopsidedness is not satisfied and my proposed Markov Chain is not irreducible, my algorithm will still generate a randomized MBP over the subset of states reachable by the initial MBP. Something else that would be useful is having an efficient algorithm to determine whether an graph satisfies this constraint. Intuitively, given large difference between k and Δ, it seems that most graphs would satisfy that constraint too.

Future work can also apply ideas from this paper to other problems.

5 Acknowledgements

I express my thanks to my professors Dr. Shuchi Chawla and Dr. Dana Moshkovitz and my teaching assistant Rojin Rezvan for their wonderful lectures that I enjoyed and learned from throughout the semester. Lastly, I have decided to make my paper accessible at: https://github.com/amitjoshi24/randomizing-lopsided-maximum-matchings

6 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.

7 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++;
}
```