

**Algorithms for Big Data**  
**Prof. John Ebenezer Augustine**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 45**  
**Graph Streaming Algorithms: Matching**

(Refer Slide Time: 00:14)

The slide is titled "Matchings" and features a blue starburst callout that reads: "Typically interested in maximum cardinality or maximum weight matchings". Below the callout, there is a bullet point: "• Matching  $M$  is a subset of the edges such that no 2 edges in the matching  $M$  share an endpoint." Another bullet point says "• E.g.". There are two graph diagrams. The first diagram shows a path graph with nodes 1, 2, 3, 4. Edges (1,2) and (3,4) are highlighted in red, representing a matching. The second diagram shows the same path graph with nodes 1, 2, 3, 4. Edges (1,2), (2,3), and (3,4) are shown. Edge (1,2) is highlighted in blue with a weight of 5 written above it. Edges (2,3) and (3,4) are shown in red with weights 3 and 4 written below them respectively. The slide also contains the IIT Madras logo in the top right and the NPTEL logo in the bottom left.

Let us look at matching's. It is 1 more problem, we want to consider. In the graph streaming context, matching is a basically a matching  $M$  is basically a subset of the set of edges such that no 2 edges in the matching  $M$  share an endpoint. So, let us look at 1 such example. So, this is a matching, this edge and this edge and now if you look  $M$  in this case,  $M$  equal to 1 comma 2 and 3 comma 4. Now, if you think about it, why this is matching? Well, it is a subset of the edges and if you look at these 2 edges they do not share an endpoint. The first edge has end points 1 and 2 the second edge has end points 3 and 4 and they do not share an end point, so matching.

Let us look at another example, this is another matching, but if you were to try and add 1 more edge then it is no longer a matching's. So, this is not a matching and typically we are interested in either maximum cardinality of the matching. So, you would not matching that are of large cardinality or in the weighted setting maximum weight matching. So, for example, this is a matching, but this is a matching of just 1 with one edge and once you have included the edge, you cannot add any other edge. So, this is not

a maximum cardinality matching. So, for this graph the real maximum cardinality matching would be the following. So, here you include, we have already saw this 1, 2 and then 3 4, but let us look at an example of maximum weight matching.

So, for example, let us say that this is 3, 4, 5 and 2 let us make that a little bit interesting let us make that 1. Now, what is the maximum weight? Matching here, if we include 3 and 1 we only get a maximum weight of 4. If we include the edge 2, 3 then also we get a maximum weight of 4. So, really the maximum weight edge, the maximum weight matching it is just that 1 edge, 1 to 3 that will give us the maximum weight of 5.

(Refer Slide Time: 03:42)

*unweighted*

**Algorithm 3: Greedy Matching**

```
1  $M \leftarrow \emptyset;$   
2 for each  $e \in S$  do  
3   If  $M \cup \{e\}$  is a matching,  $M \leftarrow M \cup \{e\};$   
4 return  $M$ 
```

This is a 2-approximation algorithm for maximum cardinality matching.

**Why?**

NPTEL IIT MADRAS

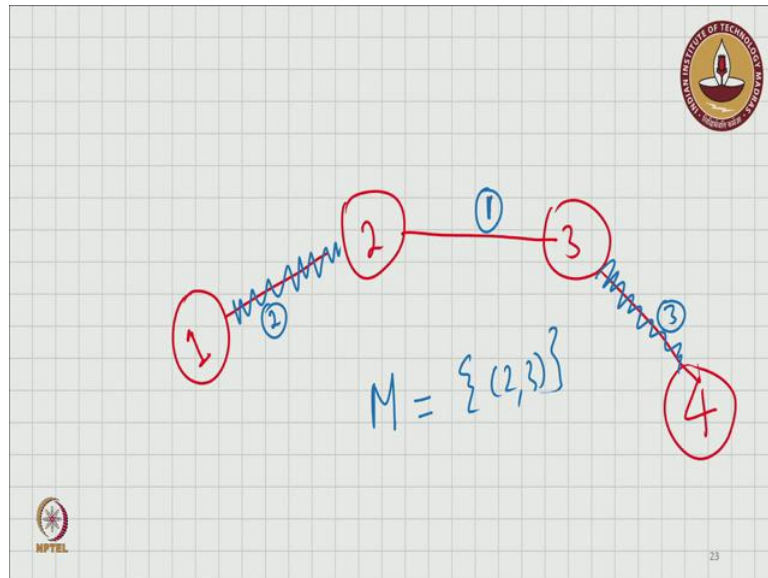
22

So, let see how we can compute the maximum cardinality matching first. So, first we are going to focus on maximum cardinality matching. It is the unweighted setting and we have do this in the streaming model. So, as usual we start with the matching being the empty set and then we ask how do, we compute the maximum cardinality matching. Now, we look at edges one at a time and for each edge we first we check whether including the edge e into the current matching retains the fact retains the matching in the problem matching property.

What is the matching property? No two edges in the matching should share an endpoint if that property is violated then you discard the set e the edge e. On the other hand, if that by including e you get a new matching then you simply add that edge to the matching. So, M you simply assign M to M union this edge e. So, it is a very simple greedy

algorithm and this gives us a good matching, but it is not the true maximum cardinality matching. So, it is only an approximation and simple let us look at a simple graph where this algorithm will fail to create to give you the actual maximum cardinality matching.

(Refer Slide Time: 05:44)



So, let say that the graph is of this form, you first get the edge 2 comma 3 then you get the edge 1 comma 2 and then you get the edge 3 comma 4, let say you get the edges in the sequence then your  $M$  after the first iteration will be the edge 2 comma 3 and then anytime you when you try to add start the edge 1 comma 2 it will violate the matching condition because they share that vertex 2. So, this edge 1 comma 2 will not be added then when we get the edge 3 comma 4. Again, we will discard that edges well because again and it is a shared vertex and so, the algorithm will output a matching with just 1 edge, but simply looking at this graph it is easy to see that the maximum cardinality matching is actually these 2 edges 1 comma 2 and 3 comma 4.

So, you get 2 edges in the matching. This is which means that we only could get half the number of edges and as it turns out this algorithm is guarantee to give you that sort of an approximation it is 2 approximation, which means that the number of edges output by this algorithm is going to be at least half the number of matching's in an optimal maximum cardinality matching.

(Refer Slide Time: 07:12)

*unweighted*

**Algorithm 3: Greedy Matching**

- 1  $M \leftarrow \emptyset;$
- 2 **for** each  $e \in S$  **do**
- 3    If  $M \cup \{e\}$  is a matching,  $M \leftarrow M \cup \{e\};$
- 4 **return**  $M$

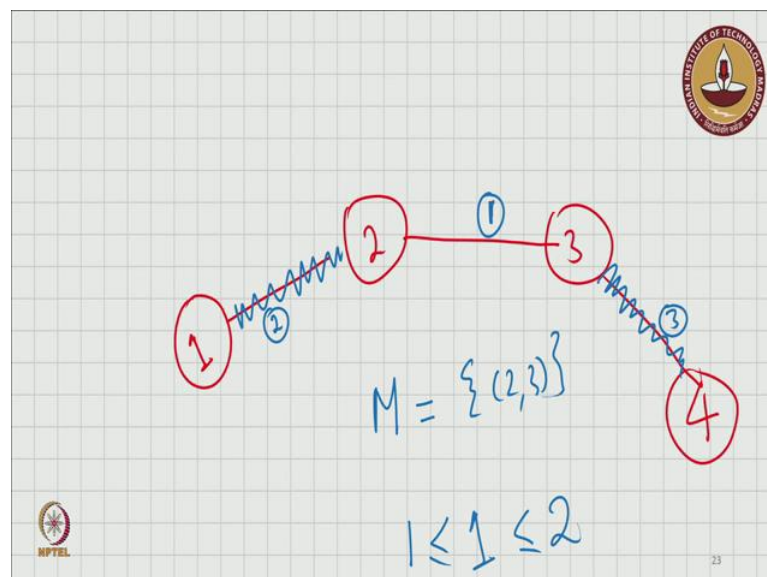
This is a 2-approximation algorithm for maximum cardinality matching.  
**Why?**

$50 \leq \leq 100$

NPTEL IIT MADRAS 22

So, if the optimal maximum cardinality matching is say 100, then we guarantee to get some number of matching's that is sandwiched within this region between 50, 100.

(Refer Slide Time: 07:57)



So, in this example the maximum cardinality matching was 2 and we got our algorithm is guarantee to output some number of matching's with just between 1 and 2 and particular we got 1. So, this is the maximum cardinality matching and now let us move want to weighted case.

(Refer Slide Time: 08:20)

**Algorithm 4: Greedy Weighted Matching**

- 1  $M \leftarrow \emptyset;$
- 2 **for each**  $e \in S$  **do**
- 3   Let  $C = \{e' \in M : e' \cap e \neq \emptyset\};$
- 4   If  $\frac{w(e)}{w(C)} \geq (1 + \gamma)$  then  $M \leftarrow M \cup \{e\} \setminus C;$
- 5 **return**  $M$

5.828 - approximation for some suitable  $\gamma$ .

So, here now we are looking at weighted graph and we are not looking at the cardinality, but we are looking at the maximum weight matching may be we want a maximum weight matching and the algorithm is little interesting here. You start with as usual the empty matching and we look at edges one at a time and for each edge, we look at we collect the set of edges in the current matching. So, this  $M$  is the current matching that we collect the set of edges that have a common endpoint with the current edge  $e$  is the current edge in our stream and we asked with, if there are edges in the current matching that have a common end point.

With the most reason, I mean the current edge in this stream and we collect all those edges and note the question is what the cardinality of this set  $C$  is? it is must be at most 2 because the current edge, let say is  $e$  and let say it is between this is edge  $e$  between  $u$  and  $v$ ,  $M$  is going to be a matching is a variant that this algorithm going to main thing and. So, at best these 2 in the set  $c$  could have 1 edge incident at  $u$  and another edge incident at  $v$ .

So, the set  $C$  could have cardinality of at most 2 and you look at that set  $C$  and the weight the  $w$  of  $C$  is basically the sum of the weights of those edges. So, let say this is 2 and this is 3, then  $w$  of  $C$  will be 5. We ask a whether the new edge  $e$  if the current edge  $e$  is from the stream is good enough for us to include in the matching because if we were to include  $e$  then we would have to throw away these 2 edges. So, before we include  $e$ , we



graph  $G$  is a weighted graph that has  $n$  vertices and the edges go from 1 to 2, 2 to 3, 3 to 4 and so on and the weights are  $1 + \epsilon$ . Now,  $1 + \epsilon$ ,  $1 + 2\epsilon$  and so on, you know. So, in general if you have a vertex  $i$  and  $i + 1$  the edge between them has weight  $1 + i\epsilon$ .

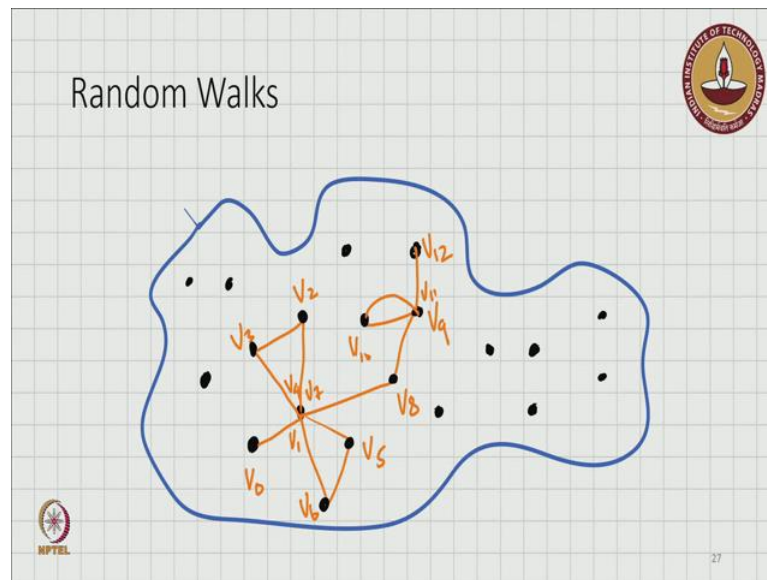
Now, let us suppose we use this condition  $w(e) > w(C)$ , then let us say that is all we have so and let us say this we get the edges in this sequence first, we get this edge, we get this second and so on. So, what will happen is the following we will first include this edge. So, in the matching and then when edge 2, 3 comes we will throw away this edge and we will then include this edge and then we will throw away this edge and we will include the next edge and so on and so forth. At the end we would have thrown away all the edges except the very last edge.

So, under this condition we will get a matching of weight  $1 + n - 1\epsilon$  as the maximum weight, but let us actually look at this a little bit more carefully see is the Best thing the highest cardinality, I mean none of the cardinality, the maximum weight matching is can actually be a lot higher. It can include this edge the previous edge will not be included, but it was basically every alternate edge can be included and let us say this edge was included in this 1 was not included and say this 1 was not included, but this 1 could have been included and so on, every alternate edge could have been included.

So, the actual maximum weight maximum weight matching is going to be much higher than what the algorithm actually will output. So, that is the output under this condition that is why we need this  $\gamma$ .



(Refer Slide Time: 16:05)



That is we wanted to discuss about matching where we have 1 last notion that we want to compute using in the streaming model, and we will this is basically a random walks and of course, we have studied random walks quite a bit in this course and you may remember recall that the random walks are actually useful to compute something called the page rank, which measures the importance of vertices in a graph and typically this is what search engines use to measure the importance of web pages in the internet.

So, random walks in that sense tend to be very, very important and fundamental to the business of a search engines and in fact, it was an algorithm that was invented by the founders of Google and which one of the key concepts that made their business. So, it is a very important notion of random, the notion in the random walks notion and the question. Now, if you have the entire graph it is easy to compute a random walk, but if all you get is the graph in this sort of a stream you get 1 edge at a time in some arbitrary order then thinks big I mean little bit tricky. So, how do you compute the random walk? And this is actually an example where your algorithm in for anything useful to be completed one pass is not enough will have to actually do multiple passes.

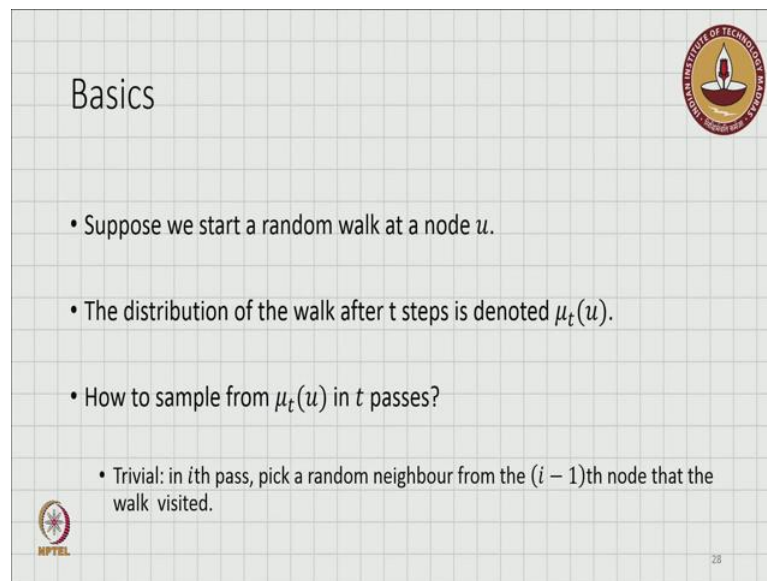
So, 1 pass is simply 1 for loop over all the edges in the graph, but can you to solve a problem like the random walks problem 1 pass is not enough you need multiple passes and, what is the random walk? Well, you started some vertex say  $v_0$  considering all of its neighbors, you choose 1 neighbor uniformly at random you have got  $v_1$  from  $v_1$  have



a bunch of neighbors, you got  $v_2$ ,  $v_3$  and then let say back to  $v_1$ , but just to just to sort of a note that this was the vertex visited forth. We are going to this sort of our random purposes label this vertex again as  $v_4$  then we got  $v_5$ ,  $v_6$  and then come back to this vertex. Now, what we call  $v_7$  then  $v_8$ ,  $v_9$ ,  $v_{10}$  and come back, we call  $v_{11}$  and  $v_{12}$ .



So, this is an example of a random walk and let us set up some basic notations.

(Refer Slide Time: 19:04)



Basics

- Suppose we start a random walk at a node  $u$ .
- The distribution of the walk after  $t$  steps is denoted  $\mu_t(u)$ .
- How to sample from  $\mu_t(u)$  in  $t$  passes?
- Trivial: in  $i$ th pass, pick a random neighbour from the  $(i - 1)$ th node that the walk visited.

28

Suppose, we start a random walk at a node  $u$  the distribution. Now, let us say we allow that random walk, to make this walk and we will allow it to do  $T$  steps after doing  $T$  steps it is these random walks is in some location and you can think about it. It is actually the location  $a$ , is a random variable, let us it can be 1 of many nodes. So, in this example that we saw after 12 steps, we reached  $v_{12}$ ;  $v_{12}$ , but if you have to run this random walk again maybe it reached type you could have reach this vertex. So, maybe it would have reached this vertex. So, this vertex or it would come back to 0 itself, there are varieties of possibilities and for each of these possibilities there is a probability associated with after 12 steps. What is the problem you can ask what is the probability that it will be the random walk will be at this vertex after 12 steps?

What is the probability of the random reaching this vertex after twelve steps? That is some another probability right and that is basically probability distribution and that we denote by  $\mu_T(u)$  it is the distribution of the random walk after  $T$  steps and now, essentially when you run a random walk for  $T$  steps you are basically sampling from this

distribution and so. Now, let us look at how we can find  $\mu_T$  of  $u$  this actually quite easy when you have  $T$  passes correct in the first pass  $u$ . Basically, let say you start at  $u$  in the first pass you look at all the neighbors of  $u$  and then after seeing all the neighbors you pick 1 of the neighbors uniformly at random.

So, the end of the first pass you get the first random walks step seems a, it is bit wasteful, but still after  $T$  passes after 1 pass you can get you can make 1 step in the random walk.

(Refer Slide Time: 21:18)

Basics

$u = (v_0, v_1, v_2) \dots$

- Suppose we start a random walk at a node  $u$ .
- The distribution of the walk after  $t$  steps is denoted  $\mu_t(u)$ .
- How to sample from  $\mu_t(u)$  in  $t$  passes?  $\sqrt{t}$  passes
- Trivial: in  $i$ th pass, pick a random neighbour from the  $(i - 1)$ th node that the walk visited.

NPTEL



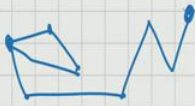
28

Let say  $u$  is nothing, but the 0th element from  $v$ ,  $v_0$  let say after the first pass, we have gone to  $v_1$ . Now, in the second pass you look at all the neighbors of  $v_1$  and simply choose 1 of them uniformly at random, you will get to the second location in the random walk sequence  $v_2$  and so on. So, you can do that for  $t$ 's  $T$  steps and you will get  $\mu_T$  of  $u$ . So, this is pretty trivial, but that is pretty wasteful and we want to do it better and in particular what we can see is we can actually sample from  $\mu_T$  of  $u$  using now  $T$  passes, but rather just square root of  $T$  passes and the idea is to try and perform square root of  $T$  random walks in parallel and somehow stitched them together.

(Refer Slide Time: 22:21)

Simplified Version

$O(n)$



1. Let  $T(v)$  be a node sampled from  $\mu_{\sqrt{t}}(v)$ .  
*( $\forall v$ , compute  $T(v)$ ). Space =  $O(n)$ .*
2. Let  $v = T^k(u) = T(\dots T(T(u)) \dots)$  where  $k$  is maximal values such that the nodes in  
$$U = \{u, T(u), T^2(u), \dots, T^{k-1}(u)\}$$
  
are all distinct and  $k \leq \sqrt{t}$ .  
*Ensuring distinctness needs more work.*

29

So, let us actually look at the algorithm. So, for simplicity we just focus on the concept rather than the details, what we are going to do is just look at a simplified version of the algorithm it is not going to be complete, but we are going to just try and understand the concept behind this. So, what we are going to do let  $T$ . So, what we have going to do is just run this algorithm for  $T$  rounds  $T$  passes square root of  $T$  passes rather, square root of  $T$  passes so, but what we are going to do is for every vertex  $v$  we are going to compute  $T$   $v$ , what is  $T$   $v$ ?  $T$   $v$  is a node sample from  $\mu$  square root of  $T$  of  $v$ .

So, if you start at  $v$  and you run the random walk for square root of let say square root of  $T$  steps then you will get you basically sample something from  $\mu$   $T$  of  $\mu$  of  $\mu$  subscripts square root of  $T$  of  $v$ , but we will do we will do this importantly we will have do this for every vertex  $v$ . So, remember we have at our disposal sum  $O$  of  $n$  space. So, we just constant space we can do one random walk starting at 1 of the vertices, but since we have  $O$  of  $n$  space, we can actually do  $n$  parallel random walks and for each 1 of them we do it square for square root  $T$  passes. So, we will get for each 1 of them you will get a square of  $T$  length random walk.

Now, the important thing is to somehow stitch these squares, these random walks to create random walk of length  $T$ .

(Refer Slide Time: 24:46)

Remarks

- Space is  $O(n + t)$ .
- Number of passes is  $O(\sqrt{t})$ .
- The ideas can be extended to computing the PageRank of a node.

Handwritten notes and diagrams:

- Diagram showing vertices  $u$ ,  $v_1$ ,  $v_2$  and sets  $T(u)$ ,  $T(v_1)$ .
- Handwritten note:  $\mu_t(u)$
- Handwritten note:  $3\sqrt{t} \dots \sqrt{t} \cdot t = t$
- Handwritten note:  $\sqrt{t}$  length r.w. starting from  $u$

Essentially what happens is the following. So, from 1 of the vertices say let say this is the  $u$  vertex  $u$  and let say this is and this is  $u$ . So, let say this is the set of all vertices from  $u$ , you get  $T$  of  $u$  this is basically square root of  $T$  length random walk starting from  $u$ . Similarly, we would have done this for all the edges. So, from  $T$   $u$  is basically again 1 of the vertices in the in the graph.

So, let say that is nothing, but say  $v_1$  the thing is this. We also have computed  $T$  of  $v_1$ . So, what we can do is we can take this random walk and stitch it with this random walk, we will get a random walk of length 2 times square root of  $T$  starting from  $u$  and we can continue this. Now,  $T$   $v_1$  may maybe that is nothing, but this vertex  $v_2$ . So, then we can take this and stitch it with the rest and then you will get random walk of length 3 times square root of  $T$ . Now, if you take square root of  $T$  such if you continue to do that you will get square root of  $T$  such random walks each of length  $T$  a square root of  $T$  stitched together. You will get  $T$  length random walk and so just by stitching these short walks we will be able to perform.

You will be able to compute, we will basically be able to sample from  $\mu_T$  starting from  $u$  this is a  $T$  length random walk. So, this requires I mean now the issue this, if in doing, if we end up. So, let say this ends up in a location and happens to be again  $v_1$ , we are in trouble because we have already used the random walk starting from  $v_1$ . So, that is a little bit of a problem, that is why in this algorithm what we are doing is we are actually

this line represents the stitching that happened, but as we are trying to stitch we should be going to distinct vertices and collecting a short walks from there.

If that is somehow not distinct, we will be running into some trouble because we cannot reuse the same short walk twice. So, that will be an issue which there is a little bit more work that can be done to avoid that issue essentially by avoiding all of this, you will I mean by overcoming those challenges what we can say is that this problem can be solved with the space of  $O$  of  $n$  plus  $T$  requiring at most  $O$  of square root of  $T$  passes with high probability and then you can use these random walks to then compute the page rank.

So, just to conclude, today we have a look at graphs streaming algorithms. We have looked at a variety of algorithms starting from very basic things like testing for connectivity, minimum spanning tree and then we looked at some way some non trivial concepts in graph algorithms like graph sparsification, random walks and things of that nature and we have seen how even the sort of restricted model of streaming were able to compute these concepts reasonably well. So, with that we will conclude our lecture on graph streaming.

Thank you.