Succinct Preferential-Attachment Graphs

Ziad Ismaili Alaoui, Namrata, Sebastian Wild[1]

University of Liverpool, United Kingdom

WG 2025, Otzenhausen, Germany

[1]University of Marburg, Germany

What is the talk about?

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs: the Barabási–Albert model.

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs: the Barabási–Albert model.
- Each graph has a certain probability p of arising through that model.

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs: the Barabási–Albert model.
- Each graph has a certain probability p of arising through that model.
- We want to design a data structure that compresses a graph generated by that model using (close to) $\lg(1/p)$ bits.

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs: the Barabási–Albert model.
- Each graph has a certain probability p of arising through that model.
- We want to design a data structure that compresses a graph generated by that model using (close to) $\lg(1/p)$ bits. **Idea:** the less probable a graph is, the more bits/space we use.

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs: the Barabási–Albert model.
- Each graph has a certain probability p of arising through that model.
- We want to design a data structure that compresses a graph generated by that model using (close to) $\lg(1/p)$ bits. **Idea:** the less probable a graph is, the more bits/space we use.
- We want the data structure to allow for efficient navigational operations. **Implication:** we will use a little bit more space,

What is the talk about?

- There is a probabilistic model that generates n-vertex graphs: the Barabási–Albert model.
- Each graph has a certain probability p of arising through that model.
- We want to design a data structure that compresses a graph generated by that model using (close to) $\lg(1/p)$ bits. **Idea:** the less probable a graph is, the more bits/space we use.
- We want the data structure to allow for efficient navigational operations. **Implication:** we will use a little bit more space, but the operations will be faster!

Preferential-attachment graphs

How does the Barabási-Albert model work?

Preferential-attachment graphs

How does the Barabási-Albert model work?

Procedure:

- We fix parameters M and $n \geq 1$.

Preferential-attachment graphs

How does the Barabási-Albert model work?

Procedure:

- We fix parameters M and $n \geq 1$.
- We create a vertex v_1.

Preferential-attachment graphs

How does the Barabási-Albert model work?

Procedure:

- We fix parameters M and $n \geq 1$.
- We create a vertex v_1.
- At each step t from 2 to n, we create one vertex v_t and direct M edges from v_t to $v_{t'}$, where $t' < t$.

Preferential-attachment graphs

How does the Barabási-Albert model work?

Procedure:

- We fix parameters M and $n \geq 1$.
- We create a vertex v_1.
- At each step t from 2 to n, we create one vertex v_t and direct M edges from v_t to $v_{t'}$, where $t' < t$.
- The probability of $v_{t'}$ to be selected as an out-neighbour of v_t is proportional to its degree right before v_t was created.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

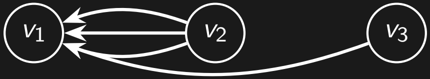
Example: fixing $n = 5$ and $M = 3$.

Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.

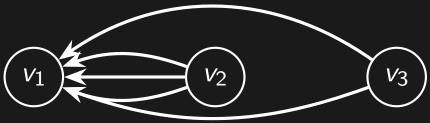
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



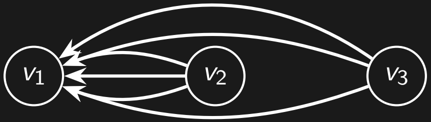
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



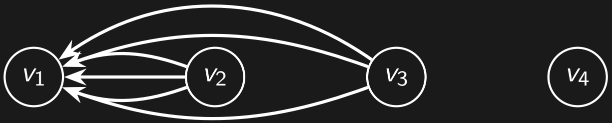
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



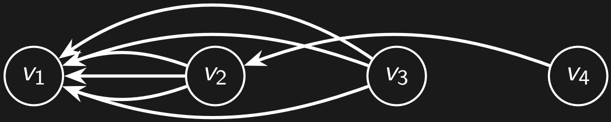
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



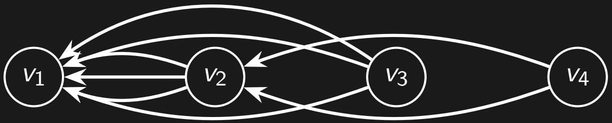
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



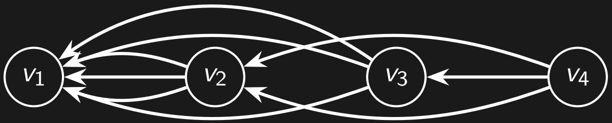
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



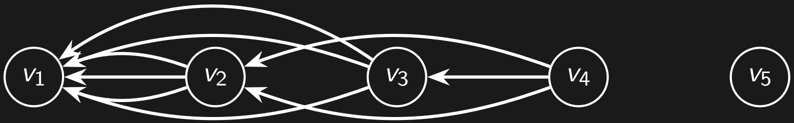
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



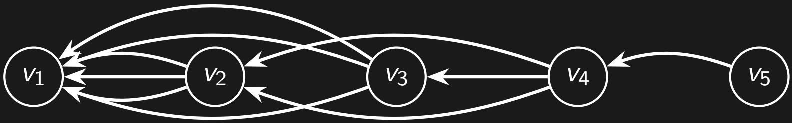
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



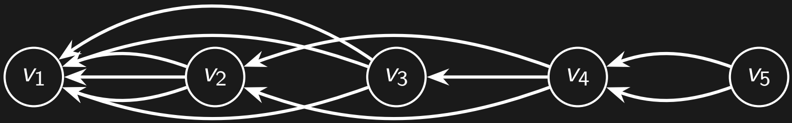
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



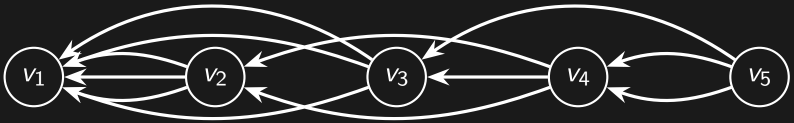
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



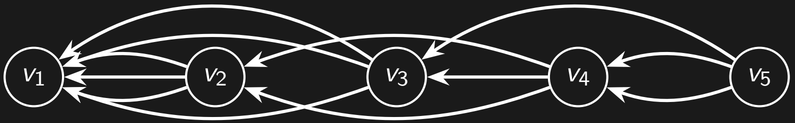
Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



Preferential-attachment graphs (example)

Example: fixing $n = 5$ and $M = 3$.



1 $\quad$ 1 $\quad$ $1 \cdot (3/6)^3$ $\quad$ $3 \cdot (3/12)^3$ $\quad$ $3 \cdot (3/18)^2 \cdot (4/18)$

Thus, the probability of the graph G_5 being generated is $\mathbb{P}[G_5] = 1/9216$.

From undirected to directed

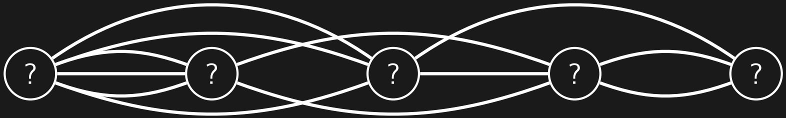
Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version.

From undirected to directed

Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version. **1-to-1 correspondence!**

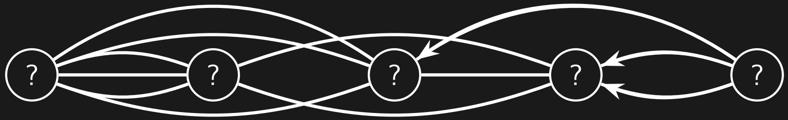
From undirected to directed

Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version. **1-to-1 correspondence!**



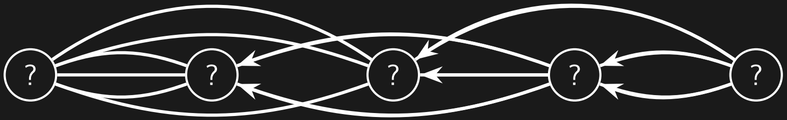
From undirected to directed

Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version. **1-to-1 correspondence!**



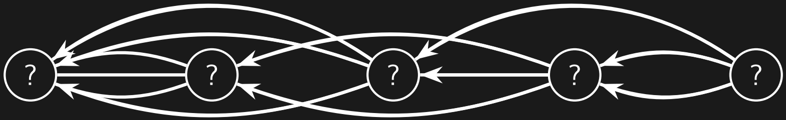
From undirected to directed

Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version. **1-to-1 correspondence!**



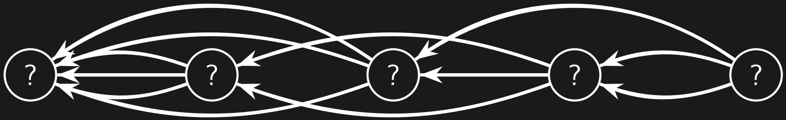
From undirected to directed

Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version. **1-to-1 correspondence!**



From undirected to directed

Assuming G is an undirected graph generated by the BA model, we can *uniquely* reconstruct its directed version. **1-to-1 correspondence!**



Towards a data structure (1/2)

Towards a data structure (1/2)

- We have a graph G (generated by the BA model).

Towards a data structure (1/2)

- We have a graph G (generated by the BA model).
- **What we want:** A data structure that compresses G and whose space usage is asymptotically instance optimal.

Towards a data structure (1/2)

- We have a graph G (generated by the BA model).
- **What we want:** A data structure that compresses G and whose space usage is asymptotically instance optimal.
- **Instance optimal** $= \lg(1/p)$ **bits.**

Towards a data structure (1/2)

- We have a graph G (generated by the BA model).
- **What we want:** A data structure that compresses G and whose space usage is asymptotically instance optimal.
- **Instance optimal** $= \lg(1/p)$ **bits.**

Definition (Adjacency string)

Let G be a directed graph.

Towards a data structure (1/2)

- We have a graph G (generated by the BA model).
- **What we want:** A data structure that compresses G and whose space usage is asymptotically instance optimal.
- **Instance optimal** $=$ $\lg(1/p)$ **bits.**

Definition (Adjacency string)

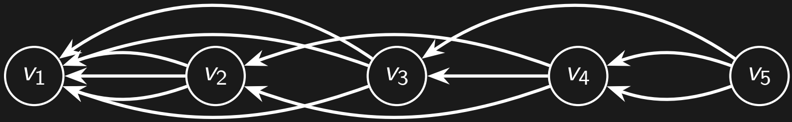
Let G be a directed graph. An *adjacency string* A of G is a string whose alphabet $\Sigma = V(G)$, and $A = N^+(v_1)N^+(v_2)\dots N^+(v_n)$ where $N^+(v)$ denotes a string concatenating the out-neighbours of v in some arbitrary order.

Towards a data structure (1/2)

- We have a graph G (generated by the BA model).
- **What we want:** A data structure that compresses G and whose space usage is asymptotically instance optimal.
- **Instance optimal** $=$ $\lg(1/p)$ **bits.**

Definition (Adjacency string)

Let G be a directed graph. An *adjacency string* A of G is a string whose alphabet $\Sigma = V(G)$, and $A = N^+(v_1)N^+(v_2)\dots N^+(v_n)$ where $N^+(v)$ denotes a string concatenating the out-neighbours of v in some arbitrary order.



$A =$ $v_1 v_1 v_1$ $\quad v_1 v_1 v_1$ $\quad v_1 v_1 v_1$ $\quad v_2 v_2 v_3$ $\quad v_4 v_3 v_4$

Towards a data structure (2/2)

Turns out... the probability p is related to the compressibility of its adjacency string A!

Towards a data structure (2/2)

Turns out... the probability p is related to the compressibility of its adjacency string A!
The *empirical entropy* of a string is a measure of compressibility.

Towards a data structure (2/2)

Turns out... the probability p is related to the compressibility of its adjacency string A! The *empirical entropy* of a string is a measure of compressibility.

Definition (Empirical Entropy)

For a string $T[1..n]$ over alphabet $\Sigma = [1..\sigma]$, the zeroth-order empirical entropy $H_0(T)$ is given by $H_0(T) = \sum_{c=1}^{\sigma} |T|_c \lg\left(\frac{n}{|T|_c}\right)$ where $|T|_c$ is the number of occurrences of the character c in T.

Key idea behind the formula: it measures how unpredictable or random the characters in a string are based on how often each character appears.

Towards a data structure (2/2)

Turns out... the probability p is related to the compressibility of its adjacency string A!
The *empirical entropy* of a string is a measure of compressibility.

Definition (Empirical Entropy)

For a string $T[1..n]$ over alphabet $\Sigma = [1..\sigma]$, the zeroth-order empirical entropy $H_0(T)$ is given by $H_0(T) = \sum_{c=1}^{\sigma} |T|_c \lg\left(\frac{n}{|T|_c}\right)$ where $|T|_c$ is the number of occurrences of the character c in T.

Key idea behind the formula: it measures how unpredictable or random the characters in a string are based on how often each character appears.

Theorem

Given an n-vertex directed graph G generated by the BA model with probability p, $\lg(1/p) = H_0(A) \pm O(nM \lg M)$.

Compressing A?

Compressing A?

- The adjacency string A contains all the information needed for structural queries.

Compressing A?

- The adjacency string A contains all the information needed for structural queries.
- A data structure that compresses A in near optimal space and allows for efficient queries on the string could be useful.

Compressing A?

- The adjacency string A contains all the information needed for structural queries.
- A data structure that compresses A in near optimal space and allows for efficient queries on the string could be useful.
- **One solution:** wavelet trees!

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$.

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

- $\mathtt{access}(S, i)$: *returns $S[i]$, the symbol at index i in S;*

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

- $\mathtt{access}(S, i)$: *returns $S[i]$, the symbol at index i in S;*
- $\mathtt{rank}_\alpha(S, i)$: *returns the number of indices with value $\alpha \in \Sigma$ in $S[1..i]$;*

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

- $\mathtt{access}(S, i)$: *returns $S[i]$, the symbol at index i in S;*
- $\mathtt{rank}_\alpha(S, i)$: *returns the number of indices with value $\alpha \in \Sigma$ in $S[1..i]$;*
- $\mathtt{select}_\alpha(S, i)$: *returns the index of the ith occurrence of value $\alpha \in \Sigma$ in S.*

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

- $\mathtt{access}(S, i)$: *returns $S[i]$, the symbol at index i in S;*
- $\mathtt{rank}_\alpha(S, i)$: *returns the number of indices with value $\alpha \in \Sigma$ in $S[1..i]$;*
- $\mathtt{select}_\alpha(S, i)$: *returns the index of the ith occurrence of value $\alpha \in \Sigma$ in S.*

A is of size nM.

Wavelet trees

Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

- $\mathtt{access}(S, i)$: *returns $S[i]$, the symbol at index i in S;*
- $\mathtt{rank}_\alpha(S, i)$: *returns the number of indices with value $\alpha \in \Sigma$ in $S[1..i]$;*
- $\mathtt{select}_\alpha(S, i)$: *returns the index of the ith occurrence of value $\alpha \in \Sigma$ in S.*

A is of size nM. So we can compress A in $H_0(A) + o(nM)$ bits while supporting the above operations in $O(\lg n)$ time

Wavelet trees

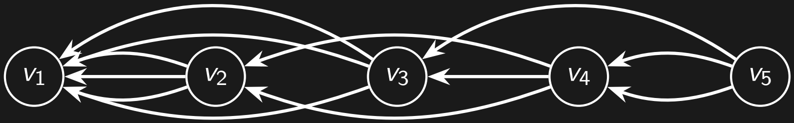
Lemma (Wavelet tree, Navarro'14)

Let S be a string of size n and alphabet $\Sigma = [1..\sigma]$. There is a data structure using $H_0(S) + o(n)$ bits that supports the following operations in $O(\lg \sigma)$ time:

- $\mathtt{access}(S, i)$: *returns $S[i]$, the symbol at index i in S;*
- $\mathtt{rank}_\alpha(S, i)$: *returns the number of indices with value $\alpha \in \Sigma$ in $S[1..i]$;*
- $\mathtt{select}_\alpha(S, i)$: *returns the index of the ith occurrence of value $\alpha \in \Sigma$ in S.*

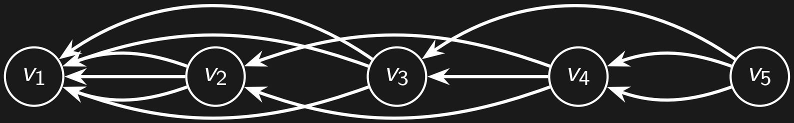
A is of size nM. So we can compress A in $H_0(A) + o(nM)$ bits while supporting the above operations in $O(\lg n)$ time (here, $\sigma = n$).

Navigational operations on G using A (1/3)



$A =$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_2 v_2 v_3$ $v_4 v_3 v_4$

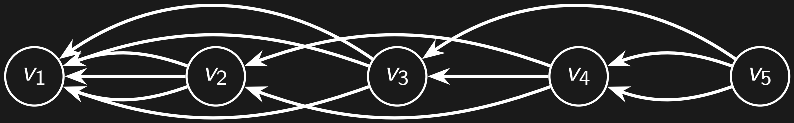
Navigational operations on G using A (1/3)



$A =$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_2 v_2 v_3$ $v_4 v_3 v_4$

Question 1. Find the jth out-neighbour of v_i [1 point].

Navigational operations on G using A (1/3)

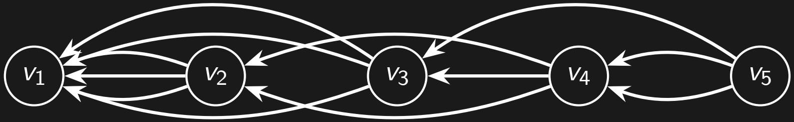


$A = \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_2 v_2 v_3 \quad v_4 v_3 v_4$

Question 1. Find the jth out-neighbour of v_i [1 point].

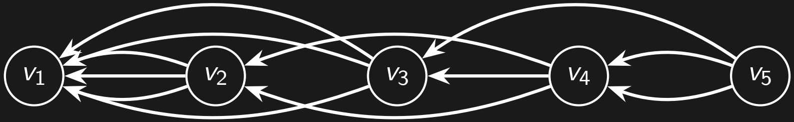
Solution: Compute $A[M(i-1)+j]$ using a $O(\lg n)$ access query on A.

Navigational operations on G using A (2/3)



$A = \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_2 v_2 v_3 \quad v_4 v_3 v_4$

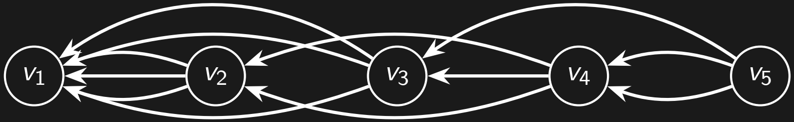
Navigational operations on G using A (2/3)



$A =$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_2 v_2 v_3$ $v_4 v_3 v_4$

Question 2. Compute the in-degree of v_i [2 points].

Navigational operations on G using A (2/3)

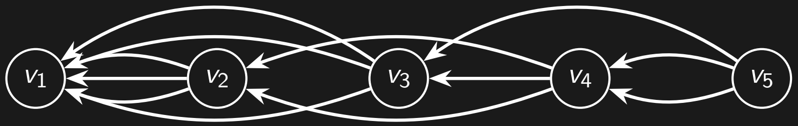


$A = \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_2 v_2 v_3 \quad v_4 v_3 v_4$

Question 2. Compute the in-degree of v_i [2 points].

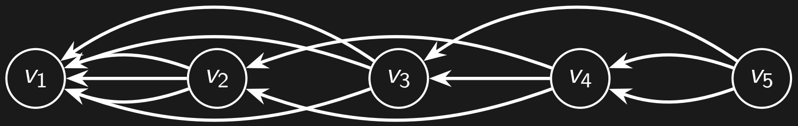
Solution: Count how many times v_i occurs in A. A simple `rank` query suffices; $O(\lg n)$.

Navigational operations on G using A (3/3)



$A =$ $v_1\, v_1\, v_1$ $v_1\, v_1\, v_1$ $v_1\, v_1\, v_1$ $v_2\, v_2\, v_3$ $v_4\, v_3\, v_4$

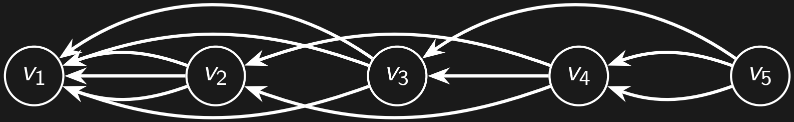
Navigational operations on G using A (3/3)



$A =$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_1 v_1 v_1$ $v_2 v_2 v_3$ $v_4 v_3 v_4$

Question 3. Find the jth in-neighbour of v_i (assume it exists) [5 points].

Navigational operations on G using A (3/3)



$$A = \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_1 v_1 v_1 \quad v_2 v_2 v_3 \quad v_4 v_3 v_4$$

Question 3. Find the jth in-neighbour of v_i (assume it exists) [5 points].

Solution: Find the position x in A where the jth occurrence of v_i is. The answer is $\lceil x/M \rceil$. The first part involves a `select` query; $O(\lg n)$.

To conclude...

To conclude...

- We have seen how to compress PA graphs generated by the BA in space asymptotically instance optimal while supporting navigational queries efficiently.

To conclude...

- We have seen how to compress PA graphs generated by the BA in space asymptotically instance optimal while supporting navigational queries efficiently.
- This scheme can be extended to general graphs (i.e. non-M-regular graphs) by using an additional vector of bits that denote the sizes of the out-neighbourhoods in A.

To conclude...

- We have seen how to compress PA graphs generated by the BA in space asymptotically instance optimal while supporting navigational queries efficiently.
- This scheme can be extended to general graphs (i.e. non-M-regular graphs) by using an additional vector of bits that denote the sizes of the out-neighbourhoods in A.
- A slightly better compression scheme can be achieved if labels (i.e. vertex identities) need not be preserved.

To conclude...

- We have seen how to compress PA graphs generated by the BA in space asymptotically instance optimal while supporting navigational queries efficiently.
- This scheme can be extended to general graphs (i.e. non-M-regular graphs) by using an additional vector of bits that denote the sizes of the out-neighbourhoods in A.
- A slightly better compression scheme can be achieved if labels (i.e. vertex identities) need not be preserved.

Cheers.