Linearity versus contiguity for encoding graphs

Tien-Nam Le

ENS de Lyon

With Christophe Cresspele, Kevin Perrot,
and Thi Ha Duong Phan

November 06, 2015

Encoding graphs

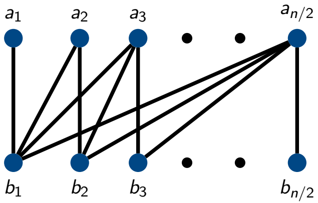


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Encoding graphs

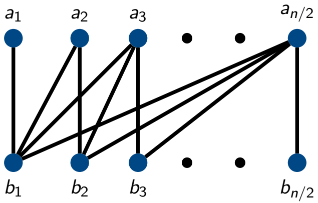


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.

Space complexity $= O(m + n)$.

Encoding graphs

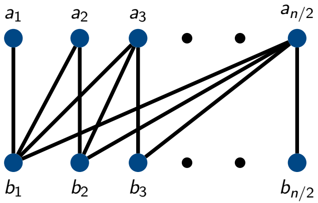


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.
Space complexity $= O(m + n)$.

Question: Can we do better?

Encoding graphs

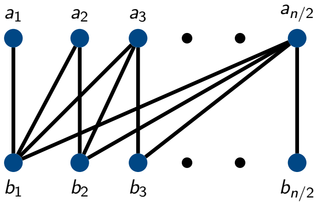


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.
Space complexity $= O(m+n)$.

Question: Can we do better?

Adjacency-intervals scheme:

- Store $\sigma(V) = (a_1, a_2, ..., a_{n/2}, b_1, b_2, ..., b_{n/2})$,

Encoding graphs

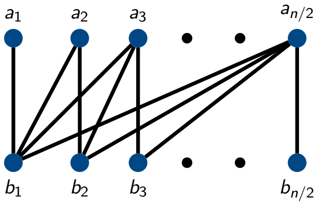


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.
Space complexity $= O(m+n)$.

Question: Can we do better?

Adjacency-intervals scheme:

- Store $\sigma(V) = (a_1, a_2, ..., a_{n/2}, b_1, b_2, ..., b_{n/2})$,
- node a_i: store b_1, b_i

Encoding graphs

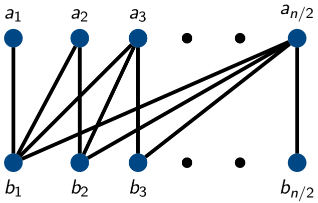


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.
Space complexity $= O(m + n)$.

Question: Can we do better?

Adjacency-intervals scheme:

- Store $\sigma(V) = (a_1, a_2, ..., a_{n/2}, b_1, b_2, ..., b_{n/2})$,
- node a_i: store b_1, b_i (represent interval $[b_1, b_2, ..., b_i]$),

Encoding graphs

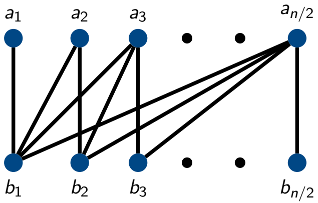


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.
Space complexity $= O(m + n)$.

Question: Can we do better?

Adjacency-intervals scheme:

- Store $\sigma(V) = (a_1, a_2, ..., a_{n/2}, b_1, b_2, ..., b_{n/2})$,
- node a_i: store b_1, b_i (represent interval $[b_1, b_2, ..., b_i]$),
- node b_j: store $a_j, a_{n/2}$ (represent interval $[a_j, a_{j+1}, ..., a_{n/2}]$).

Encoding graphs

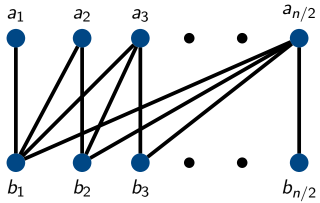


Figure: Bipartite graph on n vertices where $a_i b_j \in E \Leftrightarrow i \geq j$.

Traditional scheme: **adjacency-lists**.
Space complexity $= O(m + n)$.

Question: Can we do better?

Adjacency-intervals scheme:

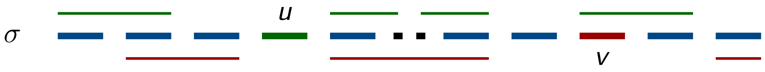
- Store $\sigma(V) = (a_1, a_2, ..., a_{n/2}, b_1, b_2, ..., b_{n/2})$,
- node a_i: store b_1, b_i (represent interval $[b_1, b_2, ..., b_i]$),
- node b_j: store $a_j, a_{n/2}$ (represent interval $[a_j, a_{j+1}, ..., a_{n/2}]$).

Space complexity $= O(n)$.

Adjacency-intervals scheme

Adjacency-intervals scheme

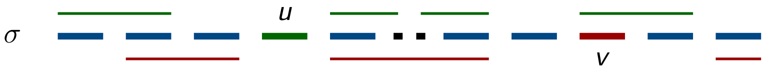
1. Store a "good" permutation $\sigma(V)$.
2. For every vertex u, store all neighbor-intervals of u in σ (store the first and last nodes).



Adjacency-intervals scheme

Adjacency-intervals scheme

1. Store a "good" permutation $\sigma(V)$.
2. For every vertex u, store all neighbor-intervals of u in σ (store the first and last nodes).



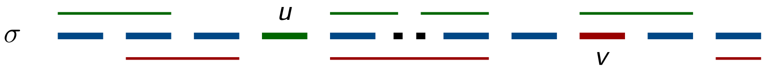
Observations:

- Complexity $\leq n + 2k_\sigma n$, where $k_\sigma = \max_u(\#\text{ intervals of } u \text{ in } \sigma)$.

Adjacency-intervals scheme

Adjacency-intervals scheme

1. Store a "good" permutation $\sigma(V)$.
2. For every vertex u, store all neighbor-intervals of u in σ (store the first and last nodes).



Observations:

- Complexity $\leq n + 2k_\sigma n$, where $k_\sigma = \max_u(\#\text{ intervals of } u \text{ in } \sigma)$.
- The smaller k_σ, the better encoding.

Adjacency-intervals scheme

Definition: contiguity of graph

$$cont(G) = \min_{\sigma} k_{\sigma}.$$

Adjacency-intervals scheme

Definition: contiguity of graph

$$cont(G) = \min_{\sigma} k_{\sigma}.$$

Observation: Every graph G on n vertices can be encoded in complexity $O\big(cont(G)n\big)$.

Adjacency-intervals scheme

Definition: contiguity of graph

$$cont(G) = \min_{\sigma} k_{\sigma}.$$

Observation: Every graph G on n vertices can be encoded in complexity $O\big(cont(G)n\big)$.

Advantages of Adjacency-intervals scheme:

- Fast encoding
- Fast querying
- Potential small space complexity.

Adjacency-intervals scheme

Definition: contiguity of graph

$$cont(G) = \min_{\sigma} k_{\sigma}.$$

Observation: Every graph G on n vertices can be encoded in complexity $O(cont(G)n)$.

Advantages of Adjacency-intervals scheme:

- Fast encoding
- Fast querying
- Potential small space complexity.

Question: Which graphs have small contiguity?

Contiguity of cographs

Theorem (Crespelle, Gambette' 2014)

- Contiguity of any **cograph** on n vertices is $O(\log n)$.

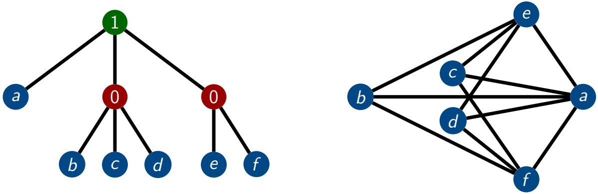


Figure: Example of a cograph and its cotree

Contiguity of cographs

Theorem (Crespelle, Gambette' 2014)

- Contiguity of any **cograph** on n vertices is $O(\log n)$.
- Contiguity of any cograph corrseponding to some **complete binary cotree** is $\Theta(\log n)$.

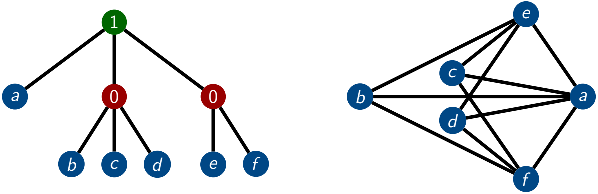
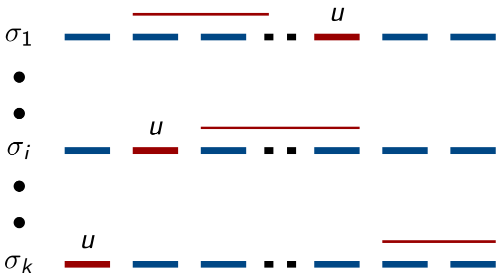


Figure: Example of a cograph and its cotree

Linearity of graph

Alternative adjacency-intervals scheme

1. Store a **collection of permutations** $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_k\}$.
2. For each $u \in V$ and $\sigma_i \in \Sigma$, store **one** neighbor-interval of u per σ_i.



Linearity of graph

Definition: Linearity

$$lin(G) = \min_{\Sigma} |\Sigma|.$$

Linearity of graph

Definition: Linearity

$$lin(G) = \min_{\Sigma} |\Sigma|.$$

Observation: Any graph G on n vertices can be encoded in complexity $O\big(lin(G)n\big)$.

Linearity of graph

Definition: Linearity

$$lin(G) = \min_{\Sigma} |\Sigma|.$$

Observation: Any graph G on n vertices can be encoded in complexity $O\big(lin(G)n\big)$.

Proporsition: $lin(G) \leq cont(G)$

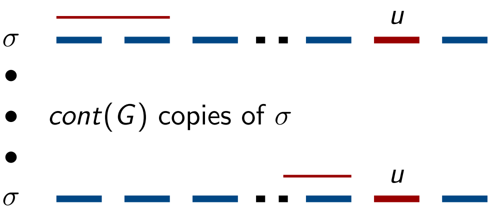
Linearity of graph

Definition: Linearity

$$lin(G) = \min_{\Sigma} |\Sigma|.$$

Observation: Any graph G on n vertices can be encoded in complexity $O\big(lin(G)n\big)$.

Proporsition: $lin(G) \leq cont(G)$



Linearity vs. contiguity

Main question

Does there exist some graph G such that $lin(G) \ll cont(G)$?

Linearity vs. contiguity

Main question

Does there exist some graph G such that $lin(G) \ll cont(G)$?

Answer: Yes!

Linearity vs. contiguity

Main question

Does there exist some graph G such that $lin(G) \ll cont(G)$?

Answer: Yes!

Main theorem (Crespelle, L., Perrot, Phan' 2015+)

Linearity of any cograph on n vertices is $O\left(\frac{\log n}{\log \log n}\right)$.

Linearity vs. contiguity

Main question

Does there exist some graph G such that $lin(G) \ll cont(G)$?

Answer: Yes!

Main theorem (Crespelle, L., Perrot, Phan' 2015+)

Linearity of any cograph on n vertices is $O\left(\frac{\log n}{\log \log n}\right)$.

Direct corollary

For any cograph G on n vertices corresponding to some complete binary cotree, $lin(G) = O\left(\frac{cont(G)}{\log \log n}\right) = o(cont(G))$.

Sketch of proof

Definition: Double factorial tree

The **double factorial tree** F^k is defined by induction:

- F^0 is a singleton.

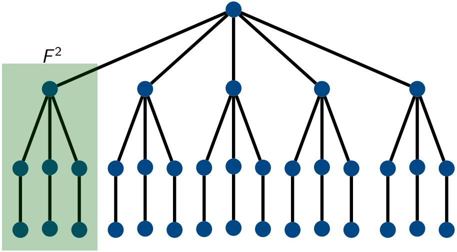


Figure: Double factorial tree F^3.

Sketch of proof

Definition: Double factorial tree

The **double factorial tree** F^k is defined by induction:

- F^0 is a singleton.
- The root of F^k has exactly $2k-1$ children, each is the root of a copy of F^{k-1}.

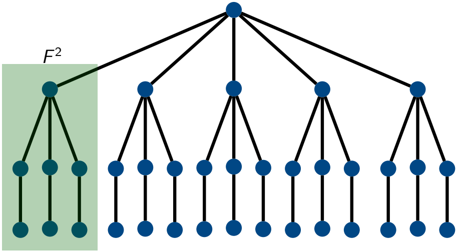


Figure: Double factorial tree F^3.

Sketch of proof

Definition: Rank

Let T be a rooted tree.

- The **rank** of T is the maximum k such that F^k is a minor of T.
- The **rank** of a node u in T is rank of subtree T_u rooted at u.

Sketch of proof

Definition: Rank

Let T be a rooted tree.

- The **rank** of T is the maximum k such that F^k is a minor of T.
- The **rank** of a node u in T is rank of subtree T_u rooted at u.

Definition: Critical node

A node u in T is **critical** if its rank is strictly greater than the rank of all its children.

Key lemma

Key lemma

Let G be a cograph whose cotree T has rank k.

(i) $lin(G) \leq 2k + 1$,

(ii) if the root is critical, then $lin(G) \leq 2k$.

Key lemma

Key lemma

Let G be a cograph whose cotree T has rank k.

(i) $lin(G) \leq 2k+1$,

(ii) if the root is critical, then $lin(G) \leq 2k$.

Proof of main theorem:

$$n = \left|V(G)\right| = \#leaves(T) \geq \#leaves(F^k) = (2k+1)!!$$

Key lemma

Key lemma

Let G be a cograph whose cotree T has rank k.

(i) $lin(G) \leq 2k+1$,

(ii) if the root is critical, then $lin(G) \leq 2k$.

Proof of main theorem:

$$n = \left|V(G)\right| = \#leaves(T) \geq \#leaves(F^k) = (2k+1)!!$$

By Stirling's approximation:

$$n \geq \frac{2\sqrt{\pi}}{e}\left(\frac{2k+2}{e}\right)^{k+1}$$

Key lemma

Key lemma

Let G be a cograph whose cotree T has rank k.

(i) $lin(G) \leq 2k+1$,

(ii) if the root is critical, then $lin(G) \leq 2k$.

Proof of main theorem:

$$n = \left|V(G)\right| = \#leaves(T) \geq \#leaves(F^k) = (2k+1)!!$$

By Stirling's approximation:

$$n \geq \frac{2\sqrt{\pi}}{e}\left(\frac{2k+2}{e}\right)^{k+1} \Longrightarrow k = O\left(\frac{\log n}{\log\log n}\right).$$

Key lemma

Key lemma

Let G be a cograph whose cotree T has rank k.

(i) $lin(G) \leq 2k+1$,

(ii) if the root is critical, then $lin(G) \leq 2k$.

Proof of main theorem:

$$n = \big|V(G)\big| = \#leaves(T) \geq \#leaves(F^k) = (2k+1)!!$$

By Stirling's approximation:

$$n \geq \frac{2\sqrt{\pi}}{e}\left(\frac{2k+2}{e}\right)^{k+1} \Longrightarrow k = O\left(\frac{\log n}{\log\log n}\right).$$

Combine with (i) in key lemma, $lin(G) = O\left(\frac{\log n}{\log\log n}\right)$. $\square$

Proof of key lemma

Prove by induction: $(ii_1) \to (i_1) \to (ii_2) \to (i_2) \to \ldots$

Proof of key lemma

Prove by induction: $(ii_1) \to (i_1) \to (ii_2) \to (i_2) \to \ldots$

Part 1. $(ii_k) \to (i_k)$:

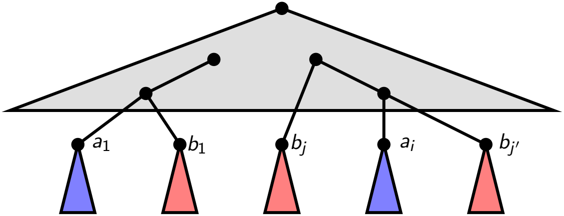


Figure: Cotree T

Proof of key lemma

Prove by induction: $(ii_1) \to (i_1) \to (ii_2) \to (i_2) \to \ldots$

Part 1. $(ii_k) \to (i_k)$: prove that G can be encoded by $2k+1$ permutations.

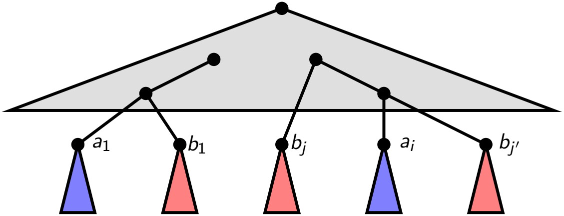


Figure: Cotree T

Proof of key lemma

Prove by induction: $(ii_1) \to (i_1) \to (ii_2) \to (i_2) \to \ldots$

Part 1. $(ii_k) \to (i_k)$: prove that G can be encoded by $2k+1$ permutations.

- $A = \{a_1, a_2, \ldots\}$: critical nodes of rank k (blue).

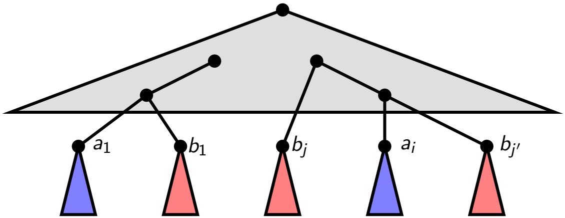


Figure: Cotree T

Proof of key lemma

Prove by induction: $(ii_1) \to (i_1) \to (ii_2) \to (i_2) \to \ldots$

Part 1. $(ii_k) \to (i_k)$: prove that G can be encoded by $2k+1$ permutations.

- $A = \{a_1, a_2, \ldots\}$: critical nodes of rank k (blue).
- $B = \{b_1, b_2, \ldots\}$: nodes of rank $k-1$, whose parent is non-critical of rank k (red).

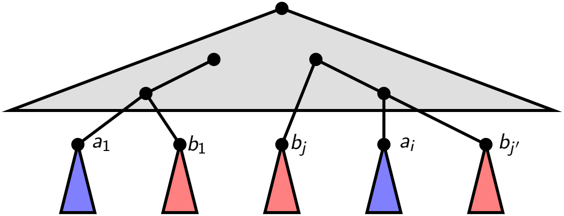


Figure: Cotree T

Proof of key lemma

Observation:

- $|A| \leq 2k$, otherwise, $rank(T) \geq k + 1$.

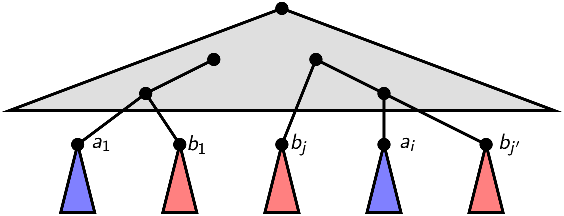


Figure: Cotree T

Proof of key lemma

Observation:

- $|A| \leq 2k$, otherwise, $rank(T) \geq k+1$.
- Although $|B|$ can be large, parent of any b_j is ancestor of some a_i.

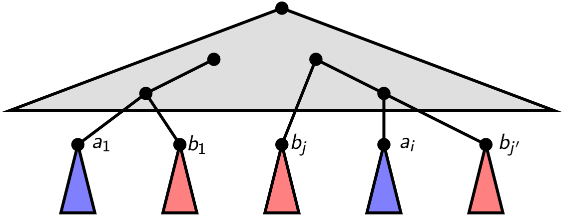


Figure: Cotree T

Proof of key lemma

Observation:

- $|A| \leq 2k$, otherwise, $rank(T) \geq k+1$.
- Although $|B|$ can be large, parent of any b_j is ancestor of some a_i.
- Contract T_{a_i} (res. T_{b_j}) into a_i (res. b_j), we get a new cotree T' of a cograph G'.

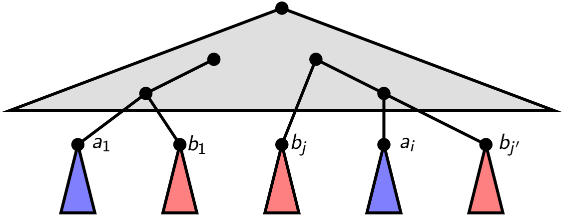


Figure: Cotree T

Proof of key lemma

Observation:

- $|A| \leq 2k$, otherwise, $rank(T) \geq k+1$.
- Although $|B|$ can be large, parent of any b_j is ancestor of some a_i.
- Contract T_{a_i} (res. T_{b_j}) into a_i (res. b_j), we get a new cotree T' of a cograph G'. G' has $|A|+|B|$ vertices, each represents a component of G.

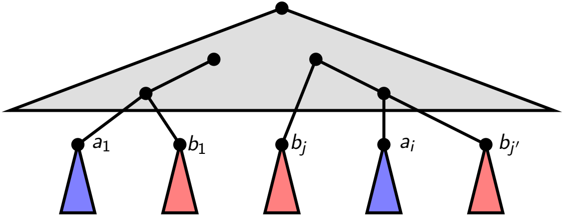


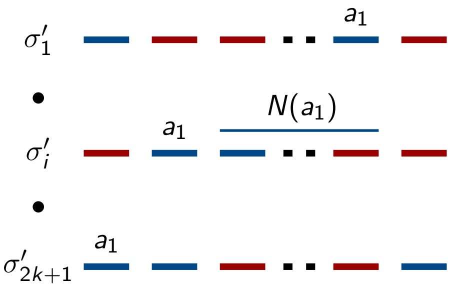
Figure: Cotree T

Proof of key lemma

Claim

There exists an encoding of G' by $\Sigma' = \{\sigma'_1, \ldots, \sigma'_{2k+1}\}$ such that:

- Neighbor set of each a_i is encoded by only **one interval**.
- Neighbor set of each b_j is encoded by at most **two intervals** in two distinct permutations.



Proof of key lemma

- Let C_{a_1} be the component of G corresponding to a_1.

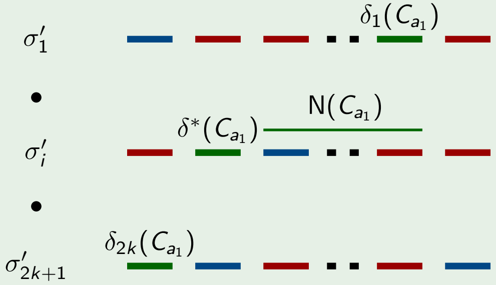
Proof of key lemma

- Let C_{a_1} be the component of G corresponding to a_1.
- a_1 is critical of rank k, so there are $\delta_1(C_{a_1}), \ldots, \delta_{2k}(C_{a_1})$ encoding C_{a_1}

Proof of key lemma

- Let C_{a_1} be the component of G corresponding to a_1.
- a_1 is critical of rank k, so there are $\delta_1(C_{a_1}), \ldots, \delta_{2k}(C_{a_1})$ encoding C_{a_1}

Replace a_1 by $\delta_1(C_{a_1}), \ldots, \delta_{2k}(C_{a_1})$:
(Note that all vertices in C_{a_1} have the same neighbors outside C_{a_1}).



Proof of key lemma

- Repeat the process for all a_i.

Proof of key lemma

- Repeat the process for all a_i.
- Repeat the process for all b_j, notice that by induction C_{b_j} can be encoded by $2k-1$ permutations.

Proof of key lemma

- Repeat the process for all a_i.
- Repeat the process for all b_j, notice that by induction C_{b_j} can be encoded by $2k-1$ permutations.
  - Neighbors outside C_{b_j} are encoded by 2 permutations.

Proof of key lemma

- Repeat the process for all a_i.
- Repeat the process for all b_j, notice that by induction C_{b_j} can be encoded by $2k-1$ permutations.
  - Neighbors outside C_{b_j} are encoded by 2 permutations.
  - Neighbors inside C_{b_j} are encoded by $2k-1$ others permutations.

Proof of key lemma

- Repeat the process for all a_i.
- Repeat the process for all b_j, notice that by induction C_{b_j} can be encoded by $2k-1$ permutations.
  - Neighbors outside C_{b_j} are encoded by 2 permutations.
  - Neighbors inside C_{b_j} are encoded by $2k-1$ others permutations.
- Finally, we obtains $2k+1$ permutations encoding G.

Proof of key lemma

- Repeat the process for all a_i.
- Repeat the process for all b_j, notice that by induction C_{b_j} can be encoded by $2k-1$ permutations.
  - Neighbors outside C_{b_j} are encoded by 2 permutations.
  - Neighbors inside C_{b_j} are encoded by $2k-1$ others permutations.
- Finally, we obtains $2k+1$ permutations encoding G.

Part 2. $(i_{k-1}) \to (ii_k)$: same idea.

Proof of key lemma

- Repeat the process for all a_i.
- Repeat the process for all b_j, notice that by induction C_{b_j} can be encoded by $2k - 1$ permutations.
  - Neighbors outside C_{b_j} are encoded by 2 permutations.
  - Neighbors inside C_{b_j} are encoded by $2k - 1$ others permutations.
- Finally, we obtains $2k + 1$ permutations encoding G.

Part 2. $(i_{k-1}) \to (ii_k)$: same idea.

The lemma is proved. $\square$

Perspective

Question: Can we find more graphs with small contiguity and linearity?

Perspective

Question: Can we find more graphs with small contiguity and linearity?

Question: Does there exist some graph with bigger gap between linearity and contiguity?

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Hybrid scheme

Encode G**:**

- Find a graph G' where $cont(G')$ is small,

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Hybrid scheme

Encode G:

- Find a graph G' where $cont(G')$ is small, and $|V \Delta V'|, |E \Delta E'|$ are small.

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Hybrid scheme

Encode G:

- Find a graph G' where $cont(G')$ is small, and $|V \Delta V'|, |E \Delta E'|$ are small.
- Encode G' by adjacency-intervals (long-term storage)

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Hybrid scheme

Encode G:

- Find a graph G' where $cont(G')$ is small, and $|V \Delta V'|, |E \Delta E'|$ are small.
- Encode G' by adjacency-intervals (long-term storage)
- Encode $V \Delta V'$ by list, and $E \Delta E'$ by adjacency-list (temporary storage).

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Hybrid scheme

Encode G:

- Find a graph G' where $cont(G')$ is small, and $|V \Delta V'|, |E \Delta E'|$ are small.
- Encode G' by adjacency-intervals (long-term storage)
- Encode $V \Delta V'$ by list, and $E \Delta E'$ by adjacency-list (temporary storage).

Add/remove vertices/edges:

- Update in temporary storage.

Perspective

Drawback of adjacency-interval scheme:

- Adding/removing vertices/edges at huge cost.

Hybrid scheme

Encode G:

- Find a graph G' where $cont(G')$ is small, and $|V \Delta V'|, |E \Delta E'|$ are small.
- Encode G' by adjacency-intervals (long-term storage)
- Encode $V \Delta V'$ by list, and $E \Delta E'$ by adjacency-list (temporary storage).

Add/remove vertices/edges:

- Update in temporary storage.
- When temporary storage is full, re-encode G.

Perspective

Definition:

Let f be a function of n. A graph G is **nearly f-contiguous** if there exists a graph G' such that

- $|V \Delta V' \bigcup E \Delta E'| = O(fn)$,
- $cont(G') = O(f)$.

Perspective

Definition:

Let f be a function of n. A graph G is **nearly f-contiguous** if there exists a graph G' such that

- $|V \Delta V' \bigcup E \Delta E'| = O(fn)$,
- $cont(G') = O(f)$.

Observation: Any nearly f-contiguous graph of order n can be encoded by hybrid scheme in complexity $O(fn)$.

Perspective

Definition:

Let f be a function of n. A graph G is **nearly f-contiguous** if there exists a graph G' such that

- $|V \Delta V' \bigcup E \Delta E'| = O(fn)$,
- $cont(G') = O(f)$.

Observation: Any nearly f-contiguous graph of order n can be encoded by hybrid scheme in complexity $O(fn)$.

Question: Which graphs are nearly $\log n$-contiguous?

The end

Thank you.