Queue-number of graphs with bounded tree-width
Veit Wiechert

Marcin Muszalski

Uniwersytet Jagielloński

November 8, 2018

The content of the slides taken from article
"Queue-number of graphs with bounded tree-width"
by Veit Wiechert[1].

Overview

Queue layout:

- Linear order of the vertices.

Queue/stack layouts

Queue layout:

- Linear order of the vertices.
- Assignment of the edges to queues, such that no two edges in a single queue are nested.

Queue/stack layouts

Queue layout:

- Linear order of the vertices.
- Assignment of the edges to queues, such that no two edges in a single queue are nested.

Stack layout:

- Linear order of the vertices.

Queue/stack layouts

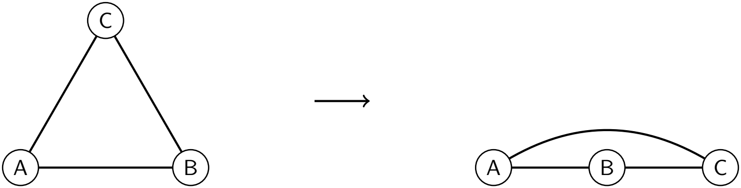
Queue layout:

- Linear order of the vertices.
- Assignment of the edges to queues, such that no two edges in a single queue are nested.

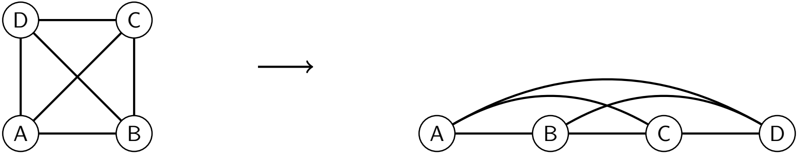
Stack layout:

- Linear order of the vertices.
- Assignment of the edges to stacks, such that no two edges in a single stack cross.

Examples

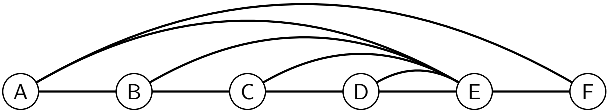


Queue-number: 1
Stack-number: 1



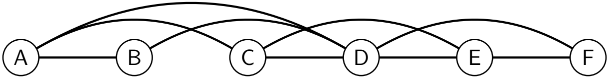
Queue-number: 2
Stack-number: 2

Examples



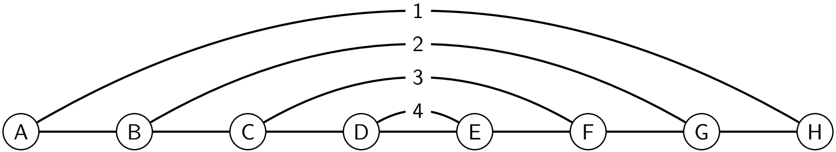
Queue-number: 2
Stack-number: 1

Examples



Queue-number: 1
Stack-number: 2

k-rainbow approach



Tree-decomposition

- $G = (V, E)$

Tree-decomposition

- $G = (V, E)$
- tree-decomposition of G is a pair $(T, \{T_x\}_{x \in V})$,
  tree T and a family of non-empty subtrees of T,

Tree-decomposition

- $G = (V, E)$
- tree-decomposition of G is a pair $(T, \{T_x\}_{x \in V})$, tree T and a family of non-empty subtrees of T, $\forall_{xy \in E} V(T_x) \cap V(T_y) \neq \emptyset$,

Tree-decomposition

- $G = (V, E)$
- tree-decomposition of G is a pair $(T, \{T_x\}_{x \in V})$,
  tree T and a family of non-empty subtrees of T,
  $\forall_{xy \in E} V(T_x) \cap V(T_y) \neq \emptyset$,
  each node $u \in V(T)$ induces a bag $\{x \in V : u \in T_x\}$.

Tree-decomposition

- $G = (V, E)$
- tree-decomposition of G is a pair $(T, \{T_x\}_{x \in V})$,
  tree T and a family of non-empty subtrees of T,
  $\forall_{xy \in E} V(T_x) \cap V(T_y) \neq \emptyset$,
  each node $u \in V(T)$ induces a bag $\{x \in V : u \in T_x\}$.
- width of the tree-decomposition:

$$\text{maximum size of a bag } - 1,$$

Tree-decomposition

- $G = (V, E)$
- tree-decomposition of G is a pair $(T, \{T_x\}_{x \in V})$, tree T and a family of non-empty subtrees of T, $\forall_{xy \in E} V(T_x) \cap V(T_y) \neq \emptyset$, each node $u \in V(T)$ induces a bag $\{x \in V : u \in T_x\}$.
- width of the tree-decomposition:

$$\text{maximum size of a bag } - 1,$$

- tree-width of G is the minimum width of a tree-decomposition of G

Tree-partition

- $G = (V, E)$
- tree-partition of G is a pair $(T, \{T_x : x \in V(T)\})$:
  - tree (forest) T,
  - partition of V into sets $\{T_x : x \in V(T)\}$, such that:

$$\forall_{uv \in V} \left(\exists!_{x \in V(T)} u, v \in T_x \veebar \exists!_{xy \in E(T)} u \in T_x \wedge v \in T_y\right),$$

- T_x is a bag of the tree-partition.

Tree-partition

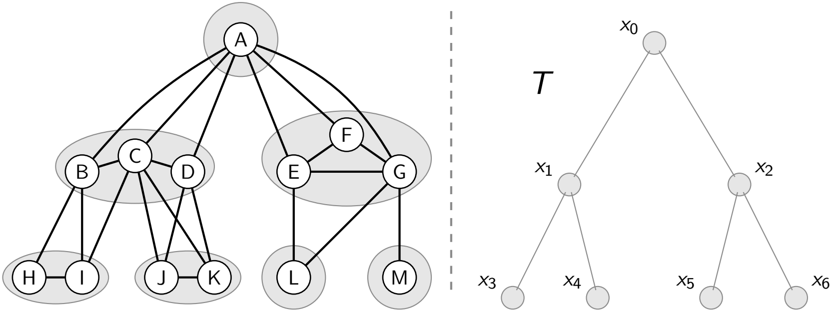


Figure: Tree-partition

k-tree

k-tree definition:

k-tree

k-tree definition:

- empty graph is a k-tree,

k-tree

k-tree definition:

- empty graph is a k-tree,
- each graph obtained by adding a vertex v to a k-tree so that the adjacent vertices of v form a clique of size at most k is also a k-tree,

Theorem 1 (Upper bound for queue-number)

Let $k \geq 0$. For all graphs G with tree-width at most k,

$$qn(G) \leq 2^k - 1.$$

Upper bound for queue-number

Lemma 2 ([Vida Dujmovi'c, Pat Morin, and David R. Wood.])

Let G be a k-tree. Then there is a rooted tree-partition $(T, \{T_x : x \in V(T)\})$ of G such that:

- *for each node x of T, the induced subgraph $G[T_x]$ is a connected $(k-1)$-tree,*
- *for each nonroot node $x \in T$, if $y \in T$ is the parent node of x in T then the vertices in T_y with a neighbor in T_x form a clique.*

Upper bound for queue-number

Lemma 2 ([Vida Dujmovi'c, Pat Morin, and David R. Wood.])

Let G be a k-tree. Then there is a rooted tree-partition $(T, \{T_x : x \in V(T)\})$ of G such that:

- *for each node x of T, the induced subgraph $G[T_x]$ is a connected $(k-1)$-tree,*
- *for each nonroot node $x \in T$, if $y \in T$ is the parent node of x in T then the vertices in T_y with a neighbor in T_x form a clique.*

Theorem 3

Let $k \geq 0$. For each k-tree G, there is a queue layout using at most $t_k = 2^k - 1$ queues, such that for each $v \in V(G)$, edges with v as their right endpoint in the layout are assigned to pairwise different queues.

Proof of theorem 3

Induction by k. For $k = 0$:

- the graph G has no edges - it requires 0 queues.

Proof of theorem 3

Induction by k. For $k = 0$:

- the graph G has no edges - it requires 0 queues.

For $k \geq 1$:

- G is a connected* k-tree.
- Let $(T, \{T_x : x \in V(T)\})$ be a tree-partition of G with r as root of T [lemma 2].
- For each node* calculate a *depth* (distance to r in T).
- Construct a linear order L^G for the queue layout of G and then assign the edges to queues.

Proof of theorem 3 - How to construct L^G?

Intuition on build L^G:

- *BFS*-like procedure (by depth).
- Dynamically construct order for each depth and append it to the right of the one already produced. To do so:
  - Specify a linear order L_d^T of the nodes at depth d in T.
  - Replace each node x in L_d^T by the linear order of the layout obtained by applying induction to the $(k-1)$-tree $G[T_x]$.

Proof of theorem 3 - How to construct L^G?

At depth 0:

- Only one node in L_0^T.
- Induction on the $(k-1)$-tree $G[T_r]$ to obtain L_0^G.

Proof of theorem 3 - How to construct L^G?

At depth 0:

- Only one node in L_0^T.
- Induction on the $(k-1)$-tree $G[T_r]$ to obtain L_0^G.

At depth d we have L_{d-1}^G, L_{d-1}^T. How to construct L_d^T?

- Order the nodes according to their parent nodes (lex-BFS ordering):
- $x < y$ in $L_d^T \iff \mathit{parent}(x) < \mathit{parent}(y)$ in L_{d-1}^T.

Proof of theorem 3 - How to construct L^G?

At depth 0:

- Only one node in L^T_0.
- Induction on the $(k-1)$-tree $G[T_r]$ to obtain L^G_0.

At depth d we have L^G_{d-1}, L^T_{d-1}. How to construct L^T_d?

- Order the nodes according to their parent nodes (lex-BFS ordering):
- $x < y$ in $L^T_d \iff parent(x) < parent(y)$ in L^T_{d-1}.

What if $parent(x) = parent(y)$?

Proof of theorem 3 - How to construct L_d^T?

Suppose that $x_1, \ldots, x_l$ have the same parent y at depth $d-1$.

- Consider the cliques $C_{x_1}, \ldots C_{x_2}$ in T_y.
- For each $i \in \{1, \ldots, l\}$ let c_{x_i} be a rightmost vertex of C_{x_i}.
- Order $x_1, \ldots, x_l$ according to the positions of $c_{x_1}, \ldots, c_{x_l}$.

Proof of theorem 3 - How to construct L_d^T?

Suppose that $x_1, \ldots, x_l$ have the same parent y at depth $d-1$.

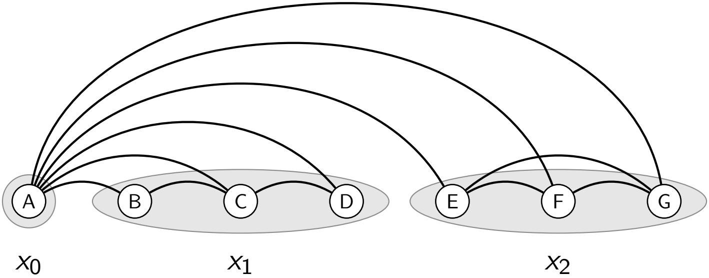
- Consider the cliques $C_{x_1}, \ldots C_{x_2}$ in T_y.
- For each $i \in \{1, \ldots, l\}$ let c_{x_i} be a rightmost vertex of C_{x_i}.
- Order $x_1, \ldots, x_l$ according to the positions of $c_{x_1}, \ldots, c_{x_l}$.

What about nodes for which $c_{x_i} = c_{x_j}$?

- Order them arbitrarily so that L_d^T becomes a linear order of nodes at depth d...

Proof of theorem 3 - How to construct L_d^T?

E.g. ordered vertices at depth at most 1 from figure 1.



Proof of theorem 3 - How to construct L^G?

By lemma 2, the bag of each node x in the tree-partition induces a $(k-1)$-tree, which allows us to apply induction.

Let L_x be the linear order of the queue layout obtained in this way.

- Replace each node x in L_d^T by the linear order L_x.
- Put the resulting order of vertices at depth d to the right of L_{d-1}^G, which yields a linear order L_d^G of all vertices at depth at most d.
- Iterate this construction until we reach the maximum depth.

Let L^G be order obtained by above procedure and L^T order of all the nodes of T.

Proof of theorem 3 - L^T

L^T has the following properties. For nodes $x, y \in V(T)$:

$$d(x) < d(y) \text{ in } T \implies x < y \text{ in } L^T \tag{1}$$

$$parent(x) < parent(y) \text{ in } L^T \implies x < y \text{ in } L^T \tag{2}$$

Property (1) asserts that L^T is a BFS ordering, and combined with property (2) we have that L^T is a Lex-BFS ordering.

- No two edges of T are nested in L^T.
- If two interbag edges uv and $u'v'$ are nested in L^G then u and u' are in the same bag of the tree-partition.

Proof of theorem 3 - coloring

First we will color the edges with colors from $\{1, \ldots, 2t_{k-1} + 1\}$ and then show that each color induces a queue with respect to L^G.

Intrabag edges:

- For each bag T_x color the contained edges according to the queue assignment that is given by the induction hypothesis for the $(k-1)$-tree $G[T_x]$
- We use the colors $1, \ldots, t_{k-1}$ for this coloring. Colors are reused.

Proof of theorem 3 - coloring

First we will color the edges with colors from $\{1, \ldots, 2t_{k-1}+1\}$ and then show that each color induces a queue with respect to L^G.

Intrabag edges:

- For each bag T_x color the contained edges according to the queue assignment that is given by the induction hypothesis for the $(k-1)$-tree $G[T_x]$
- We use the colors $1, \ldots, t_{k-1}$ for this coloring. Colors are reused.

Interbag edges, let $uv \in E(G)$ $(d(u) < d(v))$:

- there is a node x in T such that $v \in T_x$ and $u \in T_{p(x)}$,
- if $u = c_x$, then we color uv with $2t_{k-1}+1$,
- otherwise we color uv with $i + t_{k-1}$ where $i \in \{1, \ldots, t_{k-1}\}$ is the color of the intrabag edge uc_x.

Proof of theorem 3 - queues

Claim. For each color $c \in \{1, \dots, 2t_{k-1} + 1\}$, the edges of G colored c form a queue with respect to L^G.

Proof by contradiction.

- Let uv, $u'v'$ be edges with color c that are nested in L^G and $u < u' < v' < v$.
- If $c \in \{1, \dots, t_{k-1}\}$ then the edges are intrabag edges,
  - if they lie within the same bag, then they cannot be nested (valid queue layout from the induction hypothesis),
  - if they lie in different bags, then both endpoints of one edge lie before both endpoints of the other edge in L^G, a contradiction.

Proof of theorem 3 - queues

- $c \geq t_{k-1} + 1$, then uv, $u'v'$ are interbag edges.
  - By property (1) and (2) it follows that u and u' both are contained in the same bag. Let T_y be this bag and $x, x' \in V(T)$ be such that $v \in T_x$ and $v' \in T_{x'}$ ($u \in C_x$ and $u' \in C_{x'}$)

Proof of theorem 3 - queues

- $c \geq t_{k-1} + 1$, then uv, $u'v'$ are interbag edges.
  - By property (1) and (2) it follows that u and u' both are contained in the same bag. Let T_y be this bag and $x, x' \in V(T)$ be such that $v \in T_x$ and $v' \in T_{x'}$ ($u \in C_x$ and $u' \in C_{x'}$)

$c = 2t_{k-1} + 1$

- u and u' are rightmost in L^G among vertices of C_x and $C_{x'}$.
- $u = c_x$ and $u' = c_{x'}$, so $x \neq x'$,
- since x and x' share the parent y, they are ordered in L^T according to the positions of c_x, $c_{x'}$ in L^G,
- $c_x = u < u' = c_{x'}$ in L^G, this implies $x < x'$ in L^T,
- vertices of T_x lie before vertices of $t_{x'}$ in L^G, a contradiction to our assumption $v' < v$ in L^G.

Proof of theorem 3 - queues

$c \in \{t_{k-1}+1, \ldots, 2t_{k-1}\}$

- Let $i \in \{1, \ldots, t_{k-1}\}$ be such that $c = i + t_{k-1}$.
- $u \neq c_x$ and $u' \neq c_{x'}$, since $u \in C_x$ and $u' \in C_{x'}$, $u < c_x$ and $u' < c_{x'}$ in L^G.
- Edges uc_x and $u'c_{x'}$ are colored with i. This implies $c_x \neq c_{x'}$.
- By assumption that $v' < v$ in L^G we conclude $x' < x$ in L^T.
- By the fact that x and x' are ordered in L^T according to the position of c_x and $c_{x'}$ in L^G we know that $c_x < c_{x'}$ in L^G.
- $c_{x'}$ is the rightmost vertex of $C_{x'}$ in L^G, so $u < u' < c_{x'} < c_x$ in L^G. uc_x and $u'c_{x'}$ are nested and have the same color i.
- This is a contradiction to the fact that we colored these edges according to the queue layout obtained by the induction hypothesis.

$\square$

Proof of theorem 3

To complete induction step we need to show that for each $v \in V(G)$ no two edges with v as their right endpoint are colored with the same color. By contradiction, uv, $u'v$ have the same color c and $u < v$, $u' < v$ in L^G.

- By the induction hypothesis $c \in \{t_{k-1}, \ldots, 2t_{k-1} + 1\}$
- Let v be a node and $v \in T_x$. Then $u, u' \in C_x$
- Since c_x is the unique vertex of C_x that is connected by an edge in color $2t_{k-1} + 1$ to v, so $c \neq 2t_{k-1} + 1$
- Our coloring rule for uv, $u'v$ implies that uc_x, $u'c_x$ are colored with $c - t_{k-1} \in \{1, \ldots, t_{k-1}\}$
- As c_x is the rightmost vertex of C_x, we obtain that the intrabag edges uc_x and $u'c_x$ have the same color and the same right endpoint in L^G, which is a contradiction to the induction hypothesis.

We use $2t_{k-1} + 1 = 2(2^{k-1} - 1) + 1 = 2^k - 1$ queues in our layout of G, this completes the proof of the theorem.

□

Lower bounds for queue-number

Theorem 4 (Lower bounds)

For each $k \geq 2$, there is a k-tree with queue-number at least $k+1$.

k-queue game

Game between Alice and Bob on k-trees ($k \geq 2$), in which Bob has to build a queue-layout of the k-tree to be presented by Alice.

- Game starts with a $(k+1)$-clique and an arbitrary linear order of the vertices of this clique.
- Each round of the game consists of two moves:
  - Alice introduces a new vertex v and chooses a k-clique of the current graph to which v becomes adjacent.
  - Bob specifies the position in the current layout where v is inserted.
- Alice wins the k-queue game if Bob creates a rainbow of size $k+1$ in the layout.

k-queue game

Lemma 5

For each $k \geq 1$, there is an integer d_k such that Alice has a strategy to win the k-queue game within at most d_k rounds.

k-queue game

Lemma 5

For each $k \geq 1$, there is an integer d_k such that Alice has a strategy to win the k-queue game within at most d_k rounds.

Graph G, clique C in G. We stack on C in H by introducing a new vertex v_C and by making v_C adjacent to the vertices of C.
If a graph H' is obtained by simultaneously stacking on each k-clique of H, then we call H' the k-stack of H.
$(G_i)_{i \in \mathbb{N}}$ is a family of k-trees.

- Let G_0 be a $(k+1)$-clique,
- G_i is a k-stack of G_{i-1}
- G_i contains an intrinsic copy G'_{i-1} of G_{i-1} as an induced subgraph, which is such that G_i can be obtained by taking the k-stack of G'_{i-1}.

k-queue game

Lemma 6

Given $k \geq 2$, let d_k be as in the statement of lemma 5.
Then the queue-number of the k-tree G_{d_k} is at least $k+1$.

Lemma 6 implies Theorem 4.

k-queue game

Lemma 6

Given $k \geq 2$, let d_k be as in the statement of lemma 5.
Then the queue-number of the k-tree G_{d_k} is at least $k+1$.

Lemma 6 implies Theorem 4.

Proof. Consider variant of the k-queue game.

- Alice's move in a round of the variant consists of simultaneously stacking on each possible k-clique.
- Bob's task in this round to insert all the newly introduced vertices in the current layout.
- Lemma 5 holds for this variant.

k-queue game

By contradiction, linear order L of the vertices of G_{d_k} such that there is no rainbow of size $k+1$.
We claim that Bob can use L as an instruction to avoid rainbows of size $k+1$ during the first d_k rounds in the variant of the k-queue game.

- After i rounds, the graph built by Alice is isomorphic to G_i.
- Bob has to fix induced subgraphs $H_0, \ldots, H_{d_k}$ of G_{d_k} such that $H_{d_k} = G_{d_k}$ and such that H_{i-1} is the intrinsic copy of G_{i-1} in H_i for each $i \in \{1, \ldots, d_k\}$.
- $L|_{V(H_i)}$ is an extension of $L|_{V(H_{i-1})}$ for each $i \in \{1, \ldots, d_k\}$.
- Bob can ensure that the linear order after i rounds is equal to $L|_{V(H_i)}$.
- Applying this strategy, the linear order built after d_k rounds is equal to L.
- As L does not contain a rainbow of size $k+1$ Bob wins.
  This is a contradiction to lemma 5. $\square$

References

Veit Wiechert

On the queue-number of graphs with bounded tree-width.

The Electronic Journal of Combinatorics 24(1):1.65, 2017.
http://www.combinatorics.org/v24i1p65.

Bruce A. Reed.

Algorithmic aspects of tree width.

In Recent advances in algorithms and combinatorics, pages 85–107. New York, NY: Springer, 2003.

Vida Dujmovi'c, Pat Morin, and David R. Wood.

Layout of graphs with bounded tree-width.

SIAM J. Comput., 34(3):553–579, 2005.