# Fast Algorithms Parameterized by Clique-Width

Mamadou M. Kanté

(Joint works with B. Bergougnoux and O-J. Kwon)

June, $15^{\text{th}}$, 2017, UIB

# At the Beginning

**Theorem (Courcelle et al.)**

- Every $MSOL_2$-problem can be solved in time $f(k) \cdot n$ for graphs of tree-width $k$.
- Every $MSOL_1$-problem can be solved in time $f(k) \cdot n^3$ for graphs of clique-width $k$.

- $MSOL_2$ are formulas written with the incidence representation: edge set quantifications are allowed.
- $MSOL_1$ are formulas written with the adjacency relation : only vertex (set) quantifications.
- Hamiltonicity belongs to $MSOL_2 \setminus MSOL_1$.
- $TW(\leq k) \subseteq CW(\leq 2^{k+1})$ (essentially tight).

# And then Naturally . . .

Finer time complexity questions may appear with Mike's FPT world

1. For a given (bunch of) problem(s), what is the best $f(k)$?
2. For which problems in $MSOL_2 \setminus MSOL_1$ do we still have $f(k) \cdot poly(n)$ parametrised by clique-width?
    - Already hard examples were known in the 2000's paper: color not wanted edges in a clique (of clique-width $2$).
    - XP algorithms for several well-known problems: Hamiltonicity, Chromatic number, Domatic number, etc.

# And then Naturally . . .

Finer time complexity questions may appear with Mike's FPT world

1. For a given (bunch of) problem(s), what is the best $f(k)$?
2. For which problems in $MSOL_2 \setminus MSOL_1$ do we still have $f(k) \cdot poly(n)$ parametrised by clique-width?
    - Already hard examples were known in the 2000's paper: color not wanted edges in a clique (of clique-width $2$).
    - XP algorithms for several well-known problems: Hamiltonicity, Chromatic number, Domatic number, etc.

- For tree-width a lot of work the last 10years for better time complexity.

# And then Naturally . . .

Finer time complexity questions may appear with Mike's FPT world

1. For a given (bunch of) problem(s), what is the best $f(k)$?
2. For which problems in $MSOL_2 \setminus MSOL_1$ do we still have $f(k) \cdot poly(n)$ parametrised by clique-width?
   - Already hard examples were known in the 2000's paper: color not wanted edges in a clique (of clique-width $2$).
   - XP algorithms for several well-known problems: Hamiltonicity, Chromatic number, Domatic number, etc.

- For tree-width a lot of work the last 10years for better time complexity.
- Almost nothing for clique-width, except
  - $W$-hardness proof of Fomin et al. for some well-known problems,
  - $2^{k\log(k)}$ for Domination like problems (and few others) via rank-width.

# Problems

## Feedback vertex set

A feedback vertex set is $X \subseteq V(G)$ such that $G \setminus X$ is a forest.

## Connected locally checkable properties

Let $(\sigma, \rho)$ be (co-)finite subsets of $\mathbb{N}$. A connected $(\sigma, \rho)$-dominating set is a connected $D \subseteq V(G)$ s.t.

$$|N(x) \cap D| \in \begin{cases} \sigma & \text{if } x \in D \\ \rho & \text{if } x \notin D. \end{cases}$$

Examples. (Total) domination, $d$-domination, independent set, perfect domination, . . .

## Hamiltonian Cycle

Find a cycle covering all vertices.

# Wanted Algorithms

## Objective

Let $G$ be a graph of rank-width $k$,

- One can compute in time $2^{O(k)} \cdot n^{O(1)}$ a minimum FVS.
- Given $(\sigma, \rho)$, one can compute in time $2^{O(k \cdot d)} \cdot n^{O(1)}$ an optimum connected $(\sigma, \rho)$-dominating set, $d = d(\rho, \sigma)\}$.
- One can compute a Hamiltonian cycle in time $n^{O(k)}$.
- One can compute the chromatic number in time $n^{poly(k)}$.

Such algorithms (or better) exist when parameterized by tree-width.

# Our Result (Essentially tight under ETH)

**Theorem**

Let $G$ be a graph of ~~rank-width~~ clique-width $k$,

- One can compute in time $2^{O(k)} \cdot n^{O(1)}$ a minimum FVS.
- Given $(\sigma, \rho)$, one can compute in time $2^{O(k \cdot d)} \cdot n^{O(1)}$ an optimum connected $(\sigma, \rho)$-dominating set, $d = d(\rho, \sigma)\}$.
- One can compute a Hamiltonian cycle in time $n^{O(k)}$.

Reminder. $\mathrm{rwd}(G) \leq \mathrm{cwd}(G) \leq 2^{\mathrm{rwd}(G)+1} - 1$.

# Clique-Width

Given $G$ and $H$, and $lab_G : V(G) \to [k]$, $lab_H : V(H) \to [k]$

1. $\mathbf{1}(x)$ a graph with a single vertex $x$ labeled $1$,
2. $G \oplus H$ the disjoint union of $G$ and $H$,
3. $ren_{i \to j}(G)$ rename all $i$-vertices into $j$-vertices (no more vertex labeled $i$).
4. $add_{i,j}(G))$ add all edges between $i$-vertices and $j$-vertices (no parallel edges).

The clique-width of $G$, $\mathrm{cwd}(G)$, is the minimum $k$ such that $G$ is the value of a term from the above operations.

# Clique-Width versus Rank-Width

- No known FPT polynomial algorithm time for computing CWD

  $f(k) \cdot n^3$ known for RWD.

- RWD is based on ranks of matrices, but graph operations (complicated).

  CWD operations simple and better for algorithmic purposes.

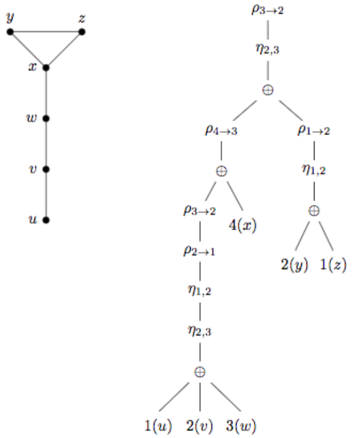
- RWD enjoys several nice structural properties, in particular links with vertex-minor quasi-ordering.

  Whilst CWD only known to be closed under induced subgraph.

# Summary

1. Hamiltonian Cycle
2. Feedback Vertex Set
3. Representatives for Weighted Partitions

# Main Idea




A path (cycle) essentially partitions the vertex set into paths in each node of the clique-width expression.
A solution at the root if a partition with one block.

# Algorithm (by Gurski)

- Consider all partitions of $V(G)$ into paths.
- A partition $\mathcal{P}$ is represented by the multiset $\{(i, j, \ell) \mid$ there are exactly $\ell$ paths between an $i$-vertex and an $j$-vertex$\}$.
- The number of such partitions is bounded by $n^{k^2}$.

$G = ren_{i \to j}(H)$. Repeat $\mathcal{P} \setminus \{(i, \ell, p), (j, \ell, p')\} \cup \{(j, \ell, p + p')\}$ until no $(i, \ell, p)$.

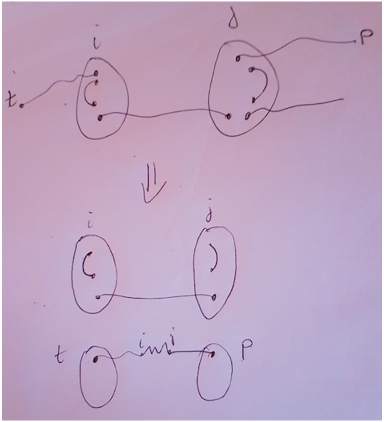
$G = G_1 \oplus G_2$. Take disjoint unions of partitions.

$G = add_{i,j}(H)$. Add as a possible partition (connect some paths)

$$\mathcal{P} \setminus \{(t, i, n_1), (j, \ell, n_2), (t, \ell, n_3)\} \cup \{(t, i, n_1 - 1), (j, \ell, n_2 - 2), (t, \ell, n_3 + 1)\}.$$

$G = add_{i,j}(H)$. Add as a possible partition (connect some paths)

$$\mathcal{P} \setminus \{(t,i,n_1),(j,\ell,n_2),(t,\ell,n_3)\} \cup \{(t,i,n_1-1),(j,\ell,n_2-2),(t,\ell,n_3+1)\}.$$



# Algorithm

- Consider all partitions of $V(G)$ into paths.
- A partition $\mathcal{P}$ is represented by the multiset $\{(i,j,\ell) \mid$ there are exactly $\ell$ paths between an $i$-vertex and an $j$-vertex$\}$.
- The number of such partitions is bounded by $n^{k^2}$.

$G = ren_{i\to j}(H)$. Repeat $\mathcal{P} \setminus \{(i,\ell,p),(j,\ell,p')\} \cup \{(j,\ell,p+p')\}$ until no $(i,\ell,p)$.

$G = G_1 \oplus G_2$. Take disjoint unions of partitions.

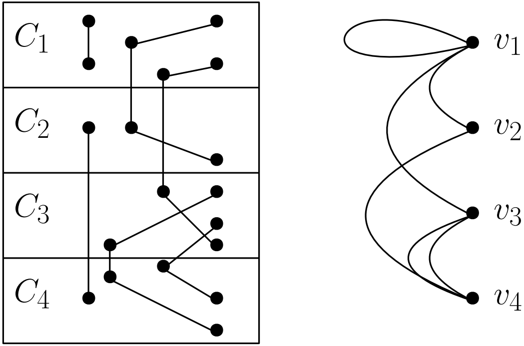
$G = add_{i,j}(H)$. Add as a possible partition (connect some paths)

$$\mathcal{P} \setminus \{(t,i,n_1),(j,\ell,n_2),(t,\ell,n_3)\} \cup \{(t,i,n_1-1),(j,\ell,n_2-2),(t,\ell,n_3+1)\}.$$

Let's turn it into an $n^{O(k)}$ one by removing unnecessary ones.

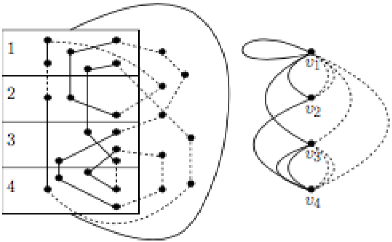
# Characteristics

- For each partition $\mathcal{P}$, compute the multigraph $Aux(\mathcal{P})$ on $k$ vertices where $\ell$ edges between $v_i$ and $v_j$ if $(i, j, \ell) \in \mathcal{P}$.
- $\mathcal{P} \equiv \mathcal{Q}$ if $Aux(\mathcal{P})$ and $Aux(\mathcal{Q})$ have same degree sequence and same connected components.



# Characteristics

- For each partition $\mathcal{P}$, compute the multigraph $Aux(\mathcal{P})$ on $k$ vertices where $\ell$ edges between $v_i$ and $v_j$ if $(i, j, \ell) \in \mathcal{P}$.
- $\mathcal{P} \equiv \mathcal{Q}$ if $Aux(\mathcal{P})$ and $Aux(\mathcal{Q})$ have same degree sequence and same connected components.




checking of Hamiltonicity is reduced to checking alternating Eulerian trail.

# Characteristics

- For each partition $\mathcal{P}$, compute the multigraph $Aux(\mathcal{P})$ on $k$ vertices where $\ell$ edges between $v_i$ and $v_j$ if $(i, j, \ell) \in \mathcal{P}$.
- $\mathcal{P} \equiv \mathcal{Q}$ if $Aux(\mathcal{P})$ and $Aux(\mathcal{Q})$ have same degree sequence and same connected components.

Reduction. Take in each equivalence class one representative.

**Proposition**

- If $\mathcal{P} \equiv \mathcal{Q}$, then $\mathcal{P}$ can be extended into a Hamiltonian iff $\mathcal{Q}$ can be.
- The operations in the algorithm preserve representativity.

Time complexity. There are $n^{O(k)}$ degree sequences and for each at most $2^{k \log(k)}$ possible partitions.

# Summary

## Compute an optimum set $D \models P$

Let $\mathcal{C}_k$ a set of characteristics classifying possible solutions.
$tab[s]$ is the optimum value for all $D$ with characteristic $s$.

1. Compute $tab[s]$ on $\mathbf{1}(x)$,
2. For each operation, compute $tab$ from $tab$'s of operands.

Assume this gives time $f(k) \cdot n^{O(1)}$, with solution in $tab[s_0]$.

# Compute an optimum set $D \models P$

Let $\mathcal{C}_k$ a set of characteristics classifying possible solutions.
$tab[s]$ is the optimum value for all $D$ with characteristic $s$.

1. Compute $tab[s]$ on $\mathbf{1}(x)$,
2. For each operation, compute $tab$ from $tab$'s of operands.

Assume this gives time $f(k) \cdot n^{O(1)}$, with solution in $tab[s_0]$.

A connected set satisfying $P$.

For each $s$, compute $\mathcal{A}[s]$ which stores the pairs $(p, w)$ s.t. there is $D$ with $p = CC(D)/[k]$ and $w := \mathrm{w}(D)$ is optimum.

## Compute an optimum set $D \models P$

Let $\mathcal{C}_k$ a set of characteristics classifying possible solutions.
$tab[s]$ is the optimum value for all $D$ with characteristic $s$.

1. Compute $tab[s]$ on $\mathbf{1}(x)$,
2. For each operation, compute $tab$ from $tab$'s of operands.

Assume this gives time $f(k) \cdot n^{O(1)}$, with solution in $tab[s_0]$.

A connected set satisfying $P$.

For each $s$, compute $\mathcal{A}[s]$ which stores the pairs $(p, w)$ s.t. there is $D$ with $p = CC(D)/[k]$ and $w := \mathrm{w}(D)$ is optimum.

The optimum connected set is $(p, w) \in \mathcal{A}[s_0]$ with $p = I$ and $I \subseteq [k]$ the set of intersected label classes in time

$$f(k) \cdot g(k) \cdot 2^{k \log(k)} \cdot n^{O(1)}.$$

# Weighted Partitions

- A weighted partition is $(p_0, p, w)$ with $(p, w) \in \Pi(V \setminus p_0) \times \mathbb{N}$.
- We let acyclic$(p, q)$ holds iff $p \sqcup q$ yields an acyclic forest, ie, $|V| + \#\mathrm{block}(p \sqcup q) - (\#\mathrm{block}(p) + \#\mathrm{block}(q)) = 0$.

# Weighted Partitions

- A weighted partition is $(p_0, p, w)$ with $(p, w) \in \Pi(V \setminus p_0) \times \mathbb{N}$.
- We let acyclic$(p, q)$ holds iff $p \sqcup q$ yields an acyclic forest, ie, $|V| + \#\text{block}(p \sqcup q) - (\#\text{block}(p) + \#\text{block}(q)) = 0$.

For $\mathcal{A}$ and $\mathcal{B}$ sets of weighted partitions

rmc$(\mathcal{A}) := \{(p_0, p, w) \in \mathcal{A} \mid \forall (p_0, p, w') \in \mathcal{A}, w' \leq w)\}$.

Remove non maximal ones.

# Weighted Partitions

- A weighted partition is $(p_0, p, w)$ with $(p, w) \in \Pi(V \setminus p_0) \times \mathbb{N}$.
- We let acyclic$(p, q)$ holds iff $p \sqcup q$ yields an acyclic forest, ie, $|V| + \#\text{block}(p \sqcup q) - (\#\text{block}(p) + \#\text{block}(q)) = 0$.

For $\mathcal{A}$ and $\mathcal{B}$ sets of weighted partitions

$$\mathsf{rmc}(\mathcal{A}) := \{(p_0, p, w) \in \mathcal{A} \mid \forall (p_0, p, w') \in \mathcal{A}, w' \leq w)\}.$$

Remove non maximal ones.

$$\mathsf{acjoin}(\mathcal{A}, \mathcal{B}) := \mathsf{rmc}\Big(\{((p_0 \setminus V') \cup (q_0 \setminus V) \cup (p_0 \cap q_0), p_{\uparrow(V' \setminus q_0)} \sqcup q_{\uparrow(V \setminus p_0)}, w_1 + w_2) \mid (p_0, p, w_1) \in \mathcal{A}, (q_0, q, w_2) \in \mathcal{B} \text{ and } \mathsf{acyclic}(p_{\uparrow(V' \setminus q_0)}, q_{\uparrow(V \setminus p_0)})\}\Big).$$

Join of two partitions

# Weighted Partitions

- A weighted partition is $(p_0, p, w)$ with $(p, w) \in \Pi(V \setminus p_0) \times \mathbb{N}$.
- We let acyclic$(p, q)$ holds iff $p \sqcup q$ yields an acyclic forest, ie, $|V| + \#\text{block}(p \sqcup q) - (\#\text{block}(p) + \#\text{block}(q)) = 0$.

For $\mathcal{A}$ and $\mathcal{B}$ sets of weighted partitions

$$\mathsf{rmc}(\mathcal{A}) := \{(p_0, p, w) \in \mathcal{A} \mid \forall (p_0, p, w') \in \mathcal{A}, w' \leq w)\}.$$

Remove non maximal ones.

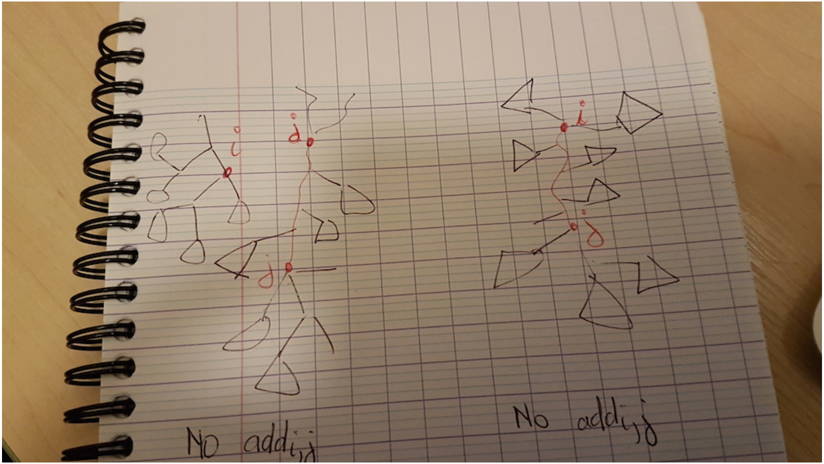
$$\mathsf{acjoin}(\mathcal{A}, \mathcal{B}) := \mathsf{rmc}\Big(\{((p_0 \setminus V') \cup (q_0 \setminus V) \cup (p_0 \cap q_0), p_{\uparrow(V' \setminus q_0)} \sqcup q_{\uparrow(V \setminus p_0)}, w_1 + w_2) \mid (p_0, p, w_1) \in \mathcal{A}, (q_0, q, w_2) \in \mathcal{B} \text{ and } \mathsf{acyclic}(p_{\uparrow(V' \setminus q_0)}, q_{\uparrow(V \setminus p_0)})\}\Big).$$

Join of two partitions

$$\mathsf{proj}(\mathcal{A}, X) := \mathsf{rmc}\Big(\{(p_0 \setminus X, p_{\Downarrow(V \setminus X)}, w) \mid (p_0, p, w) \in \mathcal{A} \text{ and } \forall p_i \in p,\ (p_i \setminus X) \neq \emptyset\}\Big).$$

For renaming, the checking of connectedness

# Needed Information



# Dynamic Programming Table

$s : [k] \to \{1, 2, -2\}$ and $V_s^+ := \{x_{i_1}, \ldots, x_{i_p}\}$ with $\{i_1, \ldots, i_p\} := s^{-1}(2)$.

**$(p_0, p, w) \in \mathcal{A}[s]$ if there is $F \subseteq G$ and $E_s^0 \subseteq \{v_0\} \times V(F)$**

1. The set $s^{-1}(1) = \{i \in [k] \mid |V(F) \cap lab_G^{-1}(i)| = 1\}$ and $p_0 = \{i \in [k] \mid |V(F) \cap lab_G^{-1}(i)| = 0\}$.
2. The graph $F^+ := (V(F) \cup \{v_0\} \cup V_s^+, E(F) \cup E_s^0 \cup E_s^+)$ is a forest where $E_s^+ := \bigcup_{1 \leq j \leq p} x_{i_j} \times (V(F) \cap lab_G^{-1}(i_j))$.
3. Each component $C$ of $(V(F) \cup \{v_0\}, E(F) \cup E_s^0)$ intersects $lab_G^{-1}(s^{-1}(\{1, 2\})) \cup \{v_0\}$.
4. The partition $p$ equals $(s^{-1}(\{1, 2\}) \cup \{v_0\}) / \sim_{F^+}$ where $i \sim j$ if a vertex in $lab_G^{-1}(i) \cap V(F)$ is connected in the graph $F^+$ to a vertex in $lab_G^{-1}(j) \cap V(F)$; we consider $lab_G^{-1}(v_0) = \{v_0\}$.

# Dynamic Programming Table

$s : [k] \to \{1, 2, -2\}$ and $V_s^+ := \{x_{i_1}, \dots, x_{i_p}\}$ with $\{i_1, \dots, i_p\} := s^{-1}(2)$.

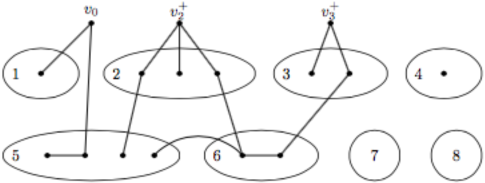



FIGURE 2. Example of graph $F^+$, here $p = \{\{v_0, 1\}, \{2, 3\}, \{4\}\}$, $s^{-1}(1) = \{1, 4\}$, $s^{-1}(2) = \{2, 3\}$, $s^{-1}(-2) = \{5, 6, 7, 8\}$ and $p_0 = \{7, 8\}$.

# $G = \mathbf{1}(x)$

$$tab_G[s] := \begin{cases} \{(\emptyset, \{\{1, v_0\}\}, \mathrm{w}(x)), (\emptyset, \{\{1\}, \{v_0\}\}, \mathrm{w}(x))\} & \text{if } s(1) = 1, \\ \{(\{1\}, \{\{v_0\}\}, 0)\} & \text{if } s(1) = -2, \\ \emptyset & \text{if } s(1) = 2. \end{cases}$$

Argue its correctness (and assuming at least two vertices in FVS).

# $G = add_{i,j}(H)$

Let $(p_0, p, w) \in \mathcal{A}_H[s']$

- If $p_0 \cap \{i, j\} \neq \emptyset$, then $(p_0, p, w) \in \mathcal{A}_G[s']$.
- Otherwise, glue blocks containing $i$ and $j$, respectively.
  - $s'(i), s'(j) \in \{1, 2\}$ (otherwise contains a cycle).
  - Let $s$ such that $s(\ell) := s'(\ell)$ for $\ell \notin \{i, j\}$ and

$$(s'(i), s'(j)) := \begin{cases} (1,1) & \text{if } s(i) = s(j) = 1 \\ (2,1) & \text{if } (s(i), s(j)) = (-2, 1) \\ (1,2) & \text{if } (s(i), s(j)) = (1, -2). \end{cases}$$

## $G = add_{i,j}(H)$

Let $(p_0, p, w) \in \mathcal{A}_H[s']$

- If $p_0 \cap \{i, j\} \neq \emptyset$, then $(p_0, p, w) \in \mathcal{A}_G[s']$.
- Otherwise, glue blocks containing $i$ and $j$, respectively.
  - $s'(i), s'(j) \in \{1, 2\}$ (otherwise contains a cycle).
  - Let $s$ such that $s(\ell) := s'(\ell)$ for $\ell \notin \{i, j\}$ and

$$(s'(i), s'(j)) := \begin{cases} (1,1) & \text{if } s(i) = s(j) = 1 \\ (2,1) & \text{if } (s(i), s(j)) = (-2, 1) \\ (1,2) & \text{if } (s(i), s(j)) = (1, -2). \end{cases}$$

Add to $\mathcal{A}_G[s]$

$\mathsf{proj}(s^{-1}(-2) \cap \{i, j\}, \mathsf{acjoin}(\{(p_0, p, w)\}, \{([k] \setminus \{i, j\}, \{\{i, j\}\}, 0)\})$.

Remark. No edge between $i$-vertices and $j$-vertices prior to $add_{i,j}$ (Irredundant expression).

# $G = G_1 \oplus G_2$

Let $(p_{01}, p_1, w_1) \in \mathcal{A}_{G_1}[s_1]$ and $(p_{02}, p_2, w_2) \in \mathcal{A}_{G_2}[s_2]$

Let $s$ such that for each $i \in [k]$

$$s(i) = \begin{cases} s_1(i) = s_2(i) = -2 & \text{if } i \in p_{01} \cap p_{02} \\ s_2(i) & \text{if } i \in p_{01}, \\ s_1(i) & \text{if } i \in p_{02}, \\ 2 & \text{if } i \in [k] \setminus (p_{01} \cup p_{02}) \text{ and } s_1(i), s_2(i) \in \{1, 2\} \\ -2 & \text{if } i \in [k] \setminus (p_{01} \cup p_{02}) \text{ and } (s_1(i), s_2(i)) \in \\ & \{(1,-2), (-2,1), (1,1), (-2,-2)\} \end{cases}$$

## $G = G_1 \oplus G_2$

Let $(p_{01}, p_1, w_1) \in \mathcal{A}_{G_1}[s_1]$ and $(p_{02}, p_2, w_2) \in \mathcal{A}_{G_2}[s_2]$

Let $s$ such that for each $i \in [k]$

$$s(i) = \begin{cases} s_1(i) = s_2(i) = -2 & \text{if } i \in p_{01} \cap p_{02} \\ s_2(i) & \text{if } i \in p_{01}, \\ s_1(i) & \text{if } i \in p_{02}, \\ 2 & \text{if } i \in [k] \setminus (p_{01} \cup p_{02}) \text{ and } s_1(i), s_2(i) \in \{1, 2\} \\ -2 & \text{if } i \in [k] \setminus (p_{01} \cup p_{02}) \text{ and } (s_1(i), s_2(i)) \in \\ & \{(1, -2), (-2, 1), (1, 1), (-2, -2)\} \end{cases}$$

Add to $\mathcal{A}_G[s]$

acjoin(proj$(s^{-1}(-2), \{(p_{01}, p_1, w_1)\})$, proj$(s^{-1}(-2), \{(p_{02}, p_2, w_2)\})$).

# $G = ren_{i \to j}(H)$

Let $(p_0, p, w) \in \mathcal{A}_H[s']$

- If $i \in p_0$, then $(p_0 \setminus \{i\}, p, w) \in \mathcal{A}_G[s']$,
- If $i \notin p_0$, but $j \in p_0$ and $s'(i) = -2$, then $(p_0 \setminus \{j\}, p, w) \in \mathcal{A}_G[s']$.

# $G = ren_{i \to j}(H)$

Let $(p_0, p, w) \in \mathcal{A}_H[s']$

- If $i \in p_0$, then $(p_0 \setminus \{i\}, p, w) \in \mathcal{A}_G[s']$,
- If $i \notin p_0$, but $j \in p_0$ and $s'(i) = -2$, then $(p_0 \setminus \{j\}, p, w) \in \mathcal{A}_G[s']$.
- If $i \notin p_0$, $j \in p_0$, $s'(i) \in \{1, 2\}$, then add to $\mathcal{A}_G[s]$ with $s(j) = s'(i)$ $\mathsf{proj}(\{i\}, \mathsf{acjoin}(\{(p_0, p, w)\}, \{([k] \setminus \{i, j\}, \{\{i, j\}\}, 0)\}))$.

# $G = ren_{i \to j}(H)$

Let $(p_0, p, w) \in \mathcal{A}_H[s']$

- If $i \in p_0$, then $(p_0 \setminus \{i\}, p, w) \in \mathcal{A}_G[s']$,
- If $i \notin p_0$, but $j \in p_0$ and $s'(i) = -2$, then $(p_0 \setminus \{j\}, p, w) \in \mathcal{A}_G[s']$.
- If $i \notin p_0$, $j \in p_0$, $s'(i) \in \{1, 2\}$, then add to $\mathcal{A}_G[s]$ with $s(j) = s'(i)$ proj$(\{i\},$ acjoin$(\{(p_0, p, w)\}, \{([k] \setminus \{i, j\}, \{\{i, j\}\}, 0)\}))$.
- Now, $i, j \notin p_0$, then
  - if $s'(i), s'(j) \in \{1, 2\}$, then add to $\mathcal{A}_G[s]$ with $s(j) = 2$ proj$(\{i\},$ acjoin$(\{(p_0, p, w)\}, \{([k] \setminus \{i, j\}, \{\{i, j\}\}, 0)\}))$.

# $G = ren_{i \to j}(H)$

Let $(p_0, p, w) \in \mathcal{A}_H[s']$

- If $i \in p_0$, then $(p_0 \setminus \{i\}, p, w) \in \mathcal{A}_G[s']$,
- If $i \notin p_0$, but $j \in p_0$ and $s'(i) = -2$, then $(p_0 \setminus \{j\}, p, w) \in \mathcal{A}_G[s']$.
- If $i \notin p_0$, $j \in p_0$, $s'(i) \in \{1, 2\}$, then add to $\mathcal{A}_G[s]$ with $s(j) = s'(i)$ proj$(\{i\},$ acjoin$(\{(p_0, p, w)\}, \{([k] \setminus \{i, j\}, \{\{i, j\}\}, 0)\}))$.
- Now, $i, j \notin p_0$, then
  - if $s'(i), s'(j) \in \{1, 2\}$, then add to $\mathcal{A}_G[s]$ with $s(j) = 2$ proj$(\{i\},$ acjoin$(\{(p_0, p, w)\}, \{([k] \setminus \{i, j\}, \{\{i, j\}\}, 0)\}))$.
  - otherwise $s'(i), s'(j) \in \{1, -2\}$, add to $\mathcal{A}_G[s]$ with $s(j) = -2$ proj$(\{i, j\}, \{(p_0, p, w)\})$.

# Summary

1. Hamiltonian Cycle
2. Feedback Vertex Set
3. Representatives for Weighted Partitions

# Ac-Representation

$\mathcal{A}$, $\mathcal{A}'$ sets of weighted partitions, and $(q_0, q, 0)$.

$\textbf{ac-opt}(\mathcal{A}, (q_0, q, 0)) := \max\{w \mid (q_0, p, w) \in \mathcal{A},\ p \sqcup q = \{V \setminus q_0\} \text{ and } \textsf{acyclic}(p, q)\}.$

# Ac-Representation

$\mathcal{A}$, $\mathcal{A}'$ sets of weighted partitions, and $(q_0, q, 0)$.

$\mathbf{ac\text{-}opt}(\mathcal{A}, (q_0, q, 0)) := \max\{w \mid (q_0, p, w) \in \mathcal{A},\ p \sqcup q = \{V \setminus q_0\} \text{ and } \mathsf{acyclic}(p, q)\}$.

$\mathcal{A}'$ ac-represents $\mathcal{A}$ if $\mathbf{ac\text{-}opt}(\mathcal{A}, (q_0, q, 0)) = \mathbf{ac\text{-}opt}(\mathcal{A}', (q_0, q, 0))$.

# Ac-Representation

$\mathcal{A}$, $\mathcal{A}'$ sets of weighted partitions, and $(q_0, q, 0)$.

$$\mathbf{ac\text{-}opt}(\mathcal{A}, (q_0, q, 0)) := \max\{w \mid (q_0, p, w) \in \mathcal{A},\ p \sqcup q = \{V \setminus q_0\} \text{ and } \textbf{acyclic}(p, q)\}.$$

$\mathcal{A}'$ ac-represents $\mathcal{A}$ if $\mathbf{ac\text{-}opt}(\mathcal{A}, (q_0, q, 0)) = \mathbf{ac\text{-}opt}(\mathcal{A}', (q_0, q, 0))$.

$f$ preserve ac-representation if $f(\mathcal{A}')$ ac-represents $f(\mathcal{A})$ if $\mathcal{A}'$ does.

# Ac-Representation

$\mathcal{A}$, $\mathcal{A}'$ sets of weighted partitions, and $(q_0, q, 0)$.

$$\textbf{ac-opt}(\mathcal{A}, (q_0, q, 0)) := \max\{w \mid (q_0, p, w) \in \mathcal{A},\ p \sqcup q = \{V \setminus q_0\} \text{ and } \textbf{acyclic}(p, q)\}.$$

$\mathcal{A}'$ ac-represents $\mathcal{A}$ if $\textbf{ac-opt}(\mathcal{A}, (q_0, q, 0)) = \textbf{ac-opt}(\mathcal{A}', (q_0, q, 0))$.

$f$ preserve ac-representation if $f(\mathcal{A}')$ ac-represents $f(\mathcal{A})$ if $\mathcal{A}'$ does.

**Proposition**

The operators rmc, proj and acjoin preserve ac-representation.

# Computing an Ac-Representative

$V' := V \cup \{v_0\}$ and cuts are tri-partitions $(U, V_1, V_2)$ with $v_0 \in V_1$

$$M[(p_0,p),(q_0,q)] := \begin{cases} 0 & \text{if } p_0 \neq q_0 \text{ or } p \sqcup q \neq \{V' \setminus p_0\}, \\ \alpha^{2|V' \setminus p_0| - (\#\text{block}(p) + \#\text{block}(q))} & \text{otherwise.} \end{cases}$$

$$C[(p_0,p),(U,V_1,V_2)] := \begin{cases} 0 & \text{if } p_0 \neq U \text{ or } p \not\sqsubseteq (V_1,V_2), \\ \alpha^{|V' \setminus U| - \#\text{block}(p)} & \text{otherwise.} \end{cases}$$

# Computing an Ac-Representative

$V' := V \cup \{v_0\}$ and cuts are tri-partitions $(U, V_1, V_2)$ with $v_0 \in V_1$

$$M[(p_0, p), (q_0, q)] := \begin{cases} 0 & \text{if } p_0 \neq q_0 \text{ or } p \sqcup q \neq \{V' \setminus p_0\}, \\ \alpha^{2|V' \setminus p_0| - (\#\text{block}(p) + \#\text{block}(q))} & \text{otherwise.} \end{cases}$$

$$C[(p_0, p), (U, V_1, V_2)] := \begin{cases} 0 & \text{if } p_0 \neq U \text{ or } p \not\sqsubseteq (V_1, V_2), \\ \alpha^{|V' \setminus U| - \#\text{block}(p)} & \text{otherwise.} \end{cases}$$

## Theorem

*We have $M = C \cdot C^t$. Moreover, there exists an algorithm* ac-reduce *that given a set of weighted partitions $\mathcal{A}$, outputs in time $|\mathcal{A}| \cdot 2^{(\omega-1)|V|} \cdot |V|^{O(1)}$ a maximum ac-generator $\mathcal{A}'$ of size $\leq (|V| + 1) \cdot 2^{|V|}$ that ac-represents $\mathcal{A}$.*

# Back to FVS Algorithm

Apply ac-reduce after computing $\mathcal{A}_G[s]$.

**Theorem**

There is an algorithm that, given an $n$-vertex graph $G$ and an irredundant clique-width $k$-expression of $G$, computes a minimum feedback vertex set in time $3^{3k} \cdot 2^{(1+\omega)k} \cdot n \cdot k^{O(1)}$.

# Back to FVS Algorithm

Apply ac-reduce after computing $\mathcal{A}_G[s]$.

**Theorem**

There is an algorithm that, given an $n$-vertex graph $G$ and an irredundant clique-width $k$-expression of $G$, computes a minimum feedback vertex set in time $3^{3k} \cdot 2^{(1+\omega)k} \cdot n \cdot k^{O(1)}$.

$G = add_{i,j}(H)$. $\mathcal{A}_G[s]$ is updated from $2$ tables in $\mathcal{A}_H$, each of size $(k+1) \cdot 2^k$, i.e., $\mathcal{A}_G$ in time $3^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{0(1)}$.

$G = ren_{i \to j}(H)$. $\mathcal{A}_G[s]$ is updated from $7$ tables in $\mathcal{A}_H$, i.e., $\mathcal{A}_G$ in time $3^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{0(1)}$.

# Back to FVS Algorithm

Apply ac-reduce after computing $\mathcal{A}_G[s]$.

## Theorem

There is an algorithm that, given an $n$-vertex graph $G$ and an irredundant clique-width $k$-expression of $G$, computes a minimum feedback vertex set in time $3^{3k} \cdot 2^{(1+\omega)k} \cdot n \cdot k^{O(1)}$.

$G = add_{i,j}(H)$. $\mathcal{A}_G[s]$ is updated from $2$ tables in $\mathcal{A}_H$, each of size $(k+1) \cdot 2^k$, i.e., $\mathcal{A}_G$ in time $3^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{0(1)}$.

$G = ren_{i \to j}(H)$. $\mathcal{A}_G[s]$ is updated from $7$ tables in $\mathcal{A}_H$, i.e., $\mathcal{A}_G$ in time $3^k \cdot 2^{(\omega+1)\cdot k} \cdot k^{0(1)}$.

$G = G_1 \oplus G_2$. $\mathcal{A}_G[s]$ is updated from $3^{2k}$ entries, each of size $2^k$, i.e. we update $\mathcal{A}_G[s]$ in time $3^{2k} \cdot 2^{(\omega+1)\cdot k}$, and $\mathcal{A}_G$ in time $3^{3k} \cdot 2^{(\omega+1)\cdot k} \cdot \overline{k^{O(1)}}$.

# Conclusion

- One can probably improve constants, but what about parametrised by rank-width?
- The algorithm for Hamiltonian cycle can be adapted for directed graph and for $q$-cycle covering.
- what about computing the chromatic number?

# Conclusion

- One can probably improve constants, but what about parametrised by rank-width?
- The algorithm for Hamiltonian cycle can be adapted for directed graph and for $q$-cycle covering.
- what about computing the chromatic number?

Let's hope fest the $70^{\text{th}}$.