Polynomial-Time Algorithms for the Subset Feedback Vertex Set Problem on Interval Graphs and Permutation Graphs

Charis Papadopoulos Spyridon Tzimas

21st International Symposium on Fundamentals of Computation Theory - FCT 2017

Bordeaux, France, September 2017

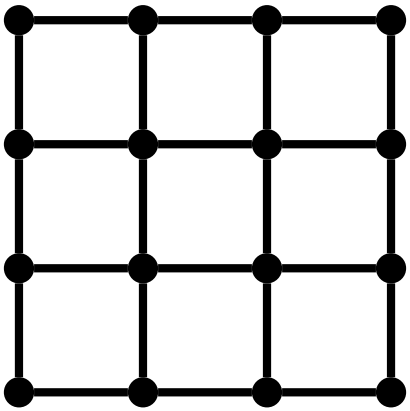
Feedback Vertex Set (FVS)

Feedback Vertex Set – FVS

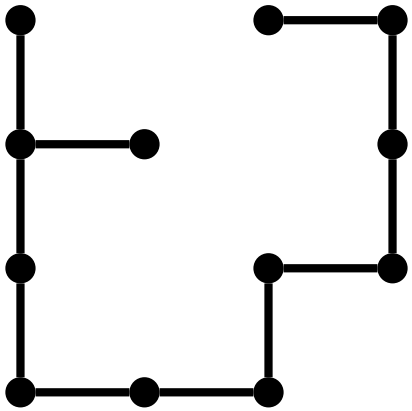
Input: A graph $G = (V, E)$

Output: Find a set $X \subset V$ of minimum cardinality such that $G - X$ is acyclic (forest).

G



$G - X$



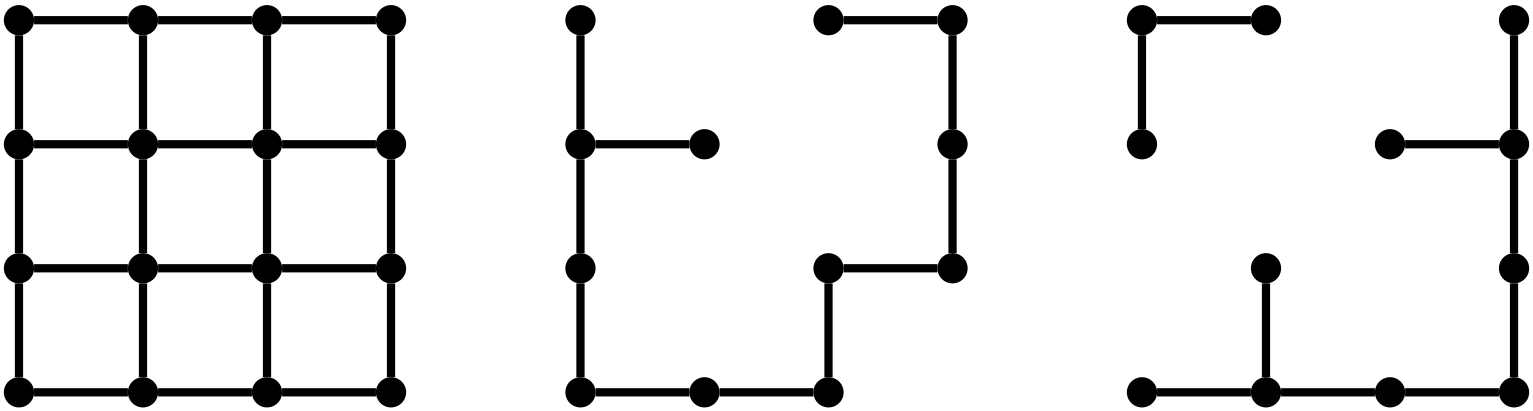
Feedback Vertex Set (FVS)

Feedback Vertex Set – FVS

Input: A graph $G = (V, E)$

Output: Find a set $X \subset V$ of minimum cardinality such that $G - X$ is acyclic (forest).

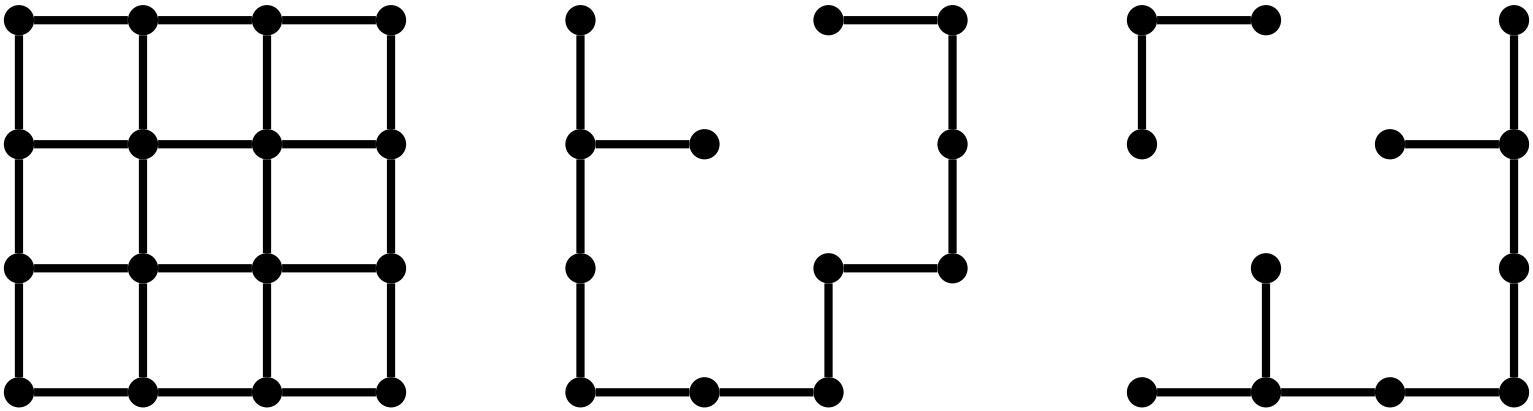
G $G - X$ $G - X'$



Feedback Vertex Set (FVS)

Feedback Vertex Set – FVS
Input: A graph $G = (V, E)$
Output: Find a set $X \subset V$ of minimum cardinality such that $G - X$ is acyclic (forest).

G $\quad$ $G - X$ $\quad$ $G - X'$



Weighted FVS:

- Weights on $V \rightarrow$ minimize $\sum_{v \in X} w(v)$

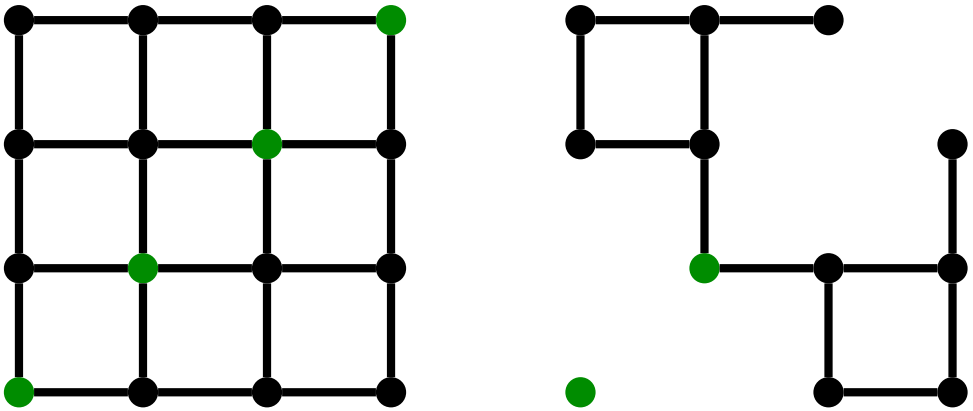
Subset Feedback Vertex Set (SFVS)

Subset Feedback Vertex Set – SFVS

Input: A graph $G = (V, E)$ and a vertex set $S \subseteq V$

Output: Find a set $X \subset V$ of minimum cardinality such that no cycle of $G - X$ contains a vertex of S.

G $\qquad$ $G - X$

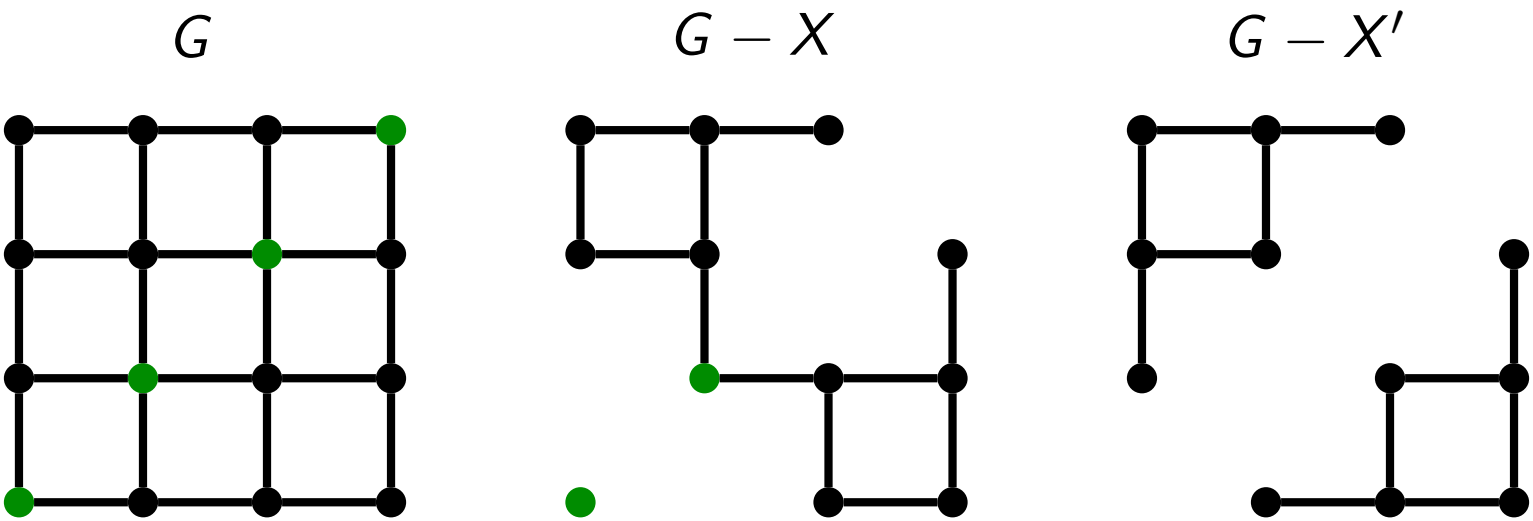


Subset Feedback Vertex Set (SFVS)

Subset Feedback Vertex Set – SFVS

Input: A graph $G = (V, E)$ and a vertex set $S \subseteq V$

Output: Find a set $X \subset V$ of minimum cardinality such that no cycle of $G - X$ contains a vertex of S.

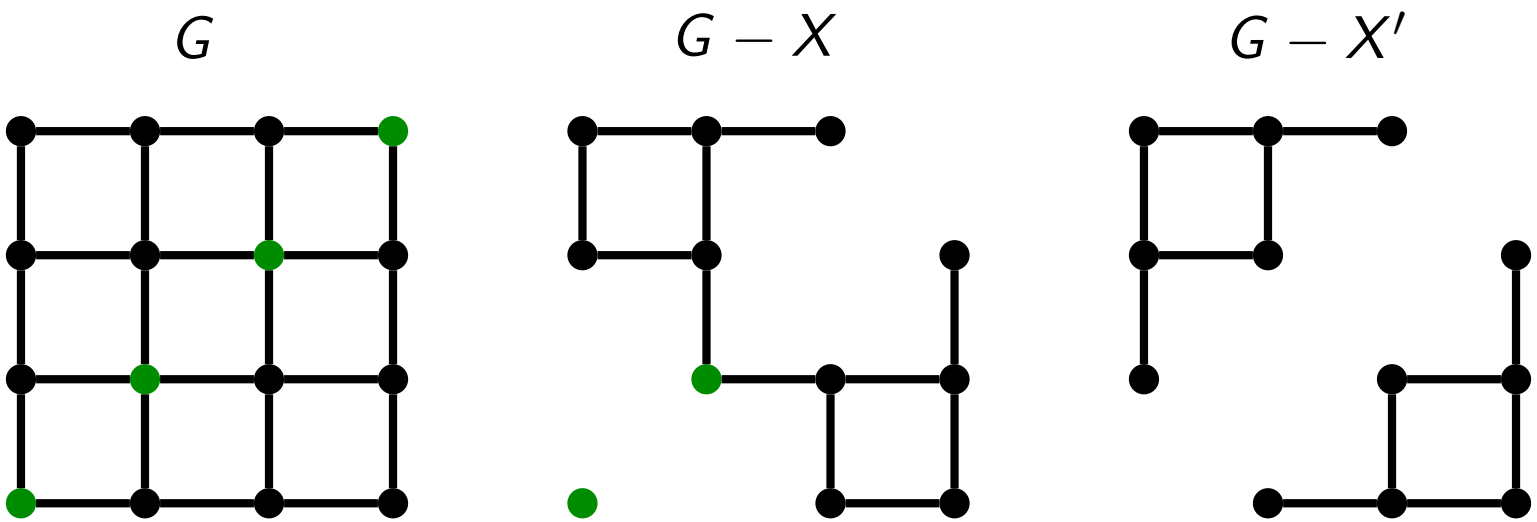


Subset Feedback Vertex Set (SFVS)

Subset Feedback Vertex Set – SFVS

Input: A graph $G = (V, E)$ and a vertex set $S \subseteq V$

Output: Find a set $X \subset V$ of minimum cardinality such that no cycle of $G - X$ contains a vertex of S.



Weighted SFVS:

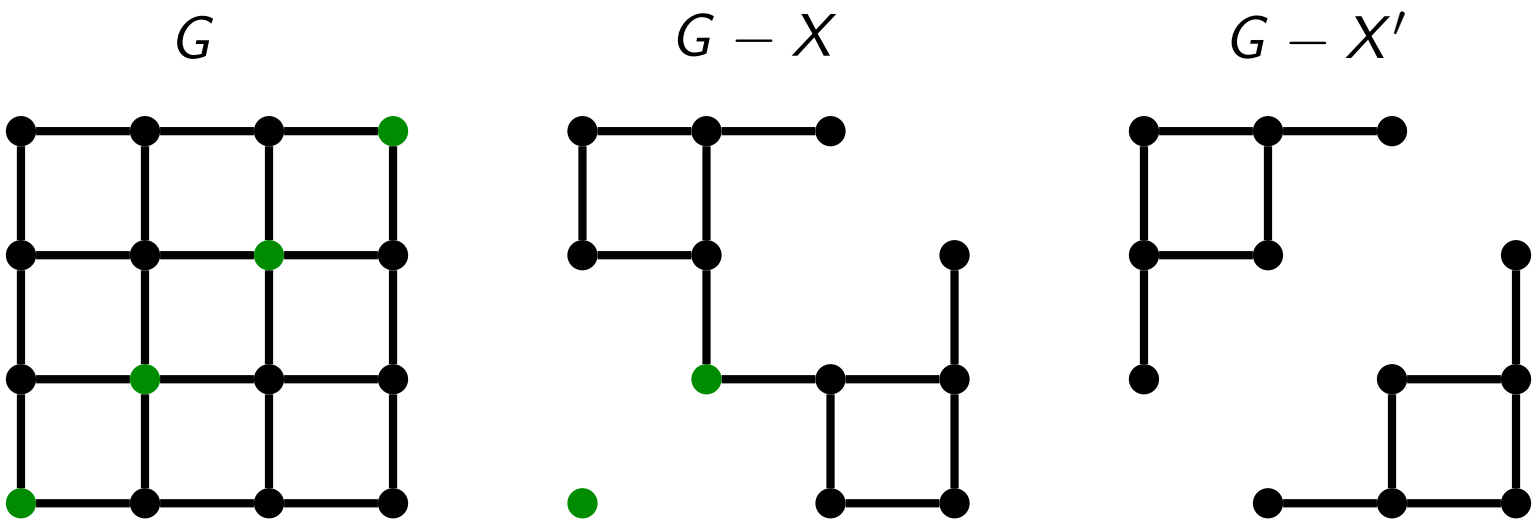
- Weights on $V \rightarrow$ minimize $\sum_{v \in X} w(v)$.

Subset Feedback Vertex Set (SFVS)

Subset Feedback Vertex Set – SFVS

Input: A graph $G = (V, E)$ and a vertex set $S \subseteq V$

Output: Find a set $X \subset V$ of minimum cardinality such that no cycle of $G - X$ contains a vertex of S.



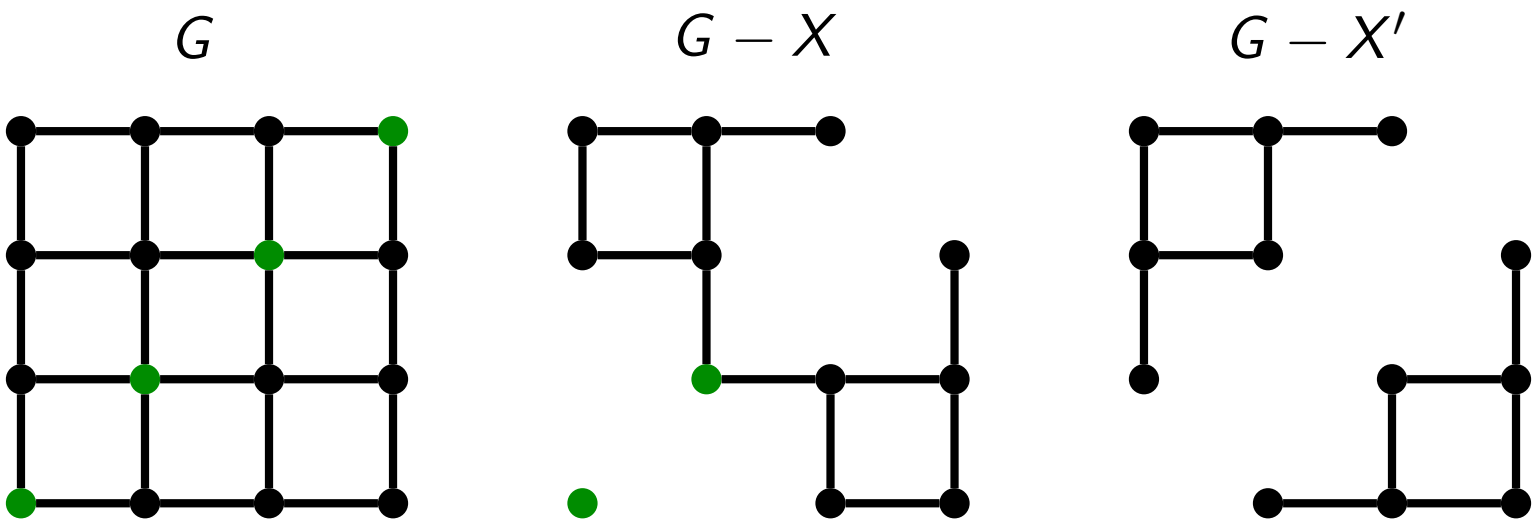
- If $S = V \Longrightarrow$ SFVS $\equiv$ FVS.

Subset Feedback Vertex Set (SFVS)

Subset Feedback Vertex Set – SFVS

Input: A graph $G = (V, E)$ and a vertex set $S \subseteq V$

Output: Find a set $X \subset V$ of minimum cardinality such that no cycle of $G - X$ contains a vertex of S.



- If $S = V \Longrightarrow$ SFVS $\equiv$ FVS.
- If $S = \{\emptyset\} \Longrightarrow X = \emptyset$.

Previous Results on FVS and SFVS

- Both problems are NP-complete (Garey and Johnson, '79)

Previous Results on FVS and SFVS

- Both problems are NP-complete (Garey and Johnson, '79)
- Exact algorithms
  - FVS: $O(1.75^n)$ (Raman et al., '08), $O(1.86^n)$ (weighted, Fomin et al., '08)
  - SFVS: $O(1.76^n)$ (Fomin et al., '16), $O(1.86^n)$ (weighted, Fomin et al., '13)
  - SFVS: chordal $O(1.68^n)$ (Golovach, '14), AT-free $O(1.62^n)$ (Chitnis, '17)

Previous Results on FVS and SFVS

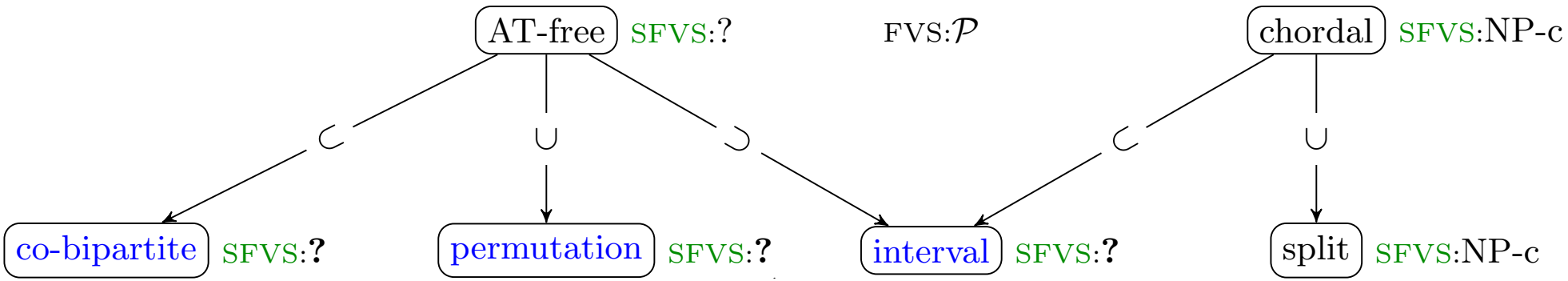
- Both problems are NP-complete (Garey and Johnson, '79)
- Exact algorithms
  - FVS: $O(1.75^n)$ (Raman et al., '08), $O(1.86^n)$ (weighted, Fomin et al., '08)
  - SFVS: $O(1.76^n)$ (Fomin et al., '16), $O(1.86^n)$ (weighted, Fomin et al., '13)
  - SFVS: chordal $O(1.68^n)$ (Golovach, '14), AT-free $O(1.62^n)$ (Chitnis, '17)
- Restricted to graph classes
  - FVS NP-complete: bipartite and planar
  - FVS $\in \mathcal{P}$: chordal (Spinrad, 2003), AT-free (Kratsch et al., 2008)

Previous Results on FVS and SFVS

- Both problems are NP-complete (Garey and Johnson, '79)
- Exact algorithms
  - FVS: $O(1.75^n)$ (Raman et al., '08), $O(1.86^n)$ (weighted, Fomin et al., '08)
  - SFVS: $O(1.76^n)$ (Fomin et al., '16), $O(1.86^n)$ (weighted, Fomin et al., '13)
  - SFVS: chordal $O(1.68^n)$ (Golovach, '14), AT-free $O(1.62^n)$ (Chitnis, '17)
- Restricted to graph classes
  - FVS NP-complete: bipartite and planar
  - FVS $\in \mathcal{P}$: chordal (Spinrad, 2003), AT-free (Kratsch et al., 2008)
  - SFVS NP-complete: split (Fomin et al., 2013)

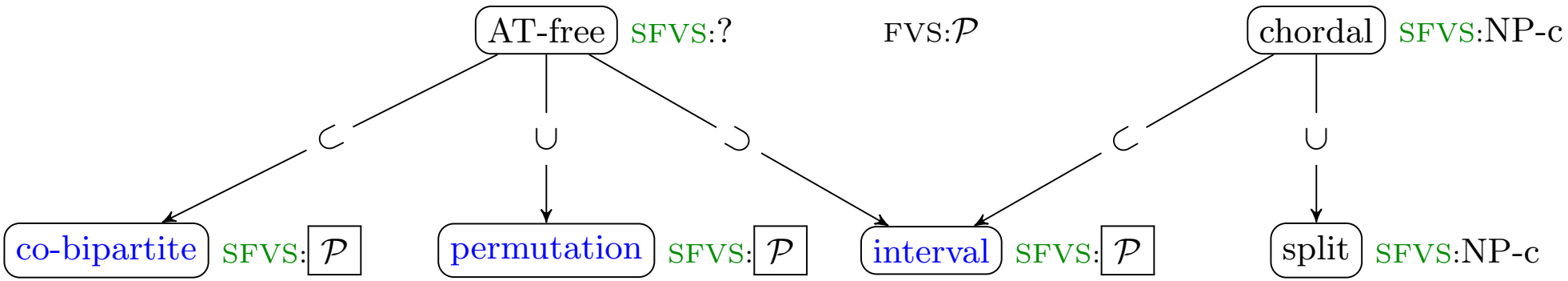
Previous Results on FVS and SFVS

- Both problems are NP-complete (Garey and Johnson, '79)
- Exact algorithms
  - FVS: $O(1.75^n)$ (Raman et al., '08), $O(1.86^n)$ (weighted, Fomin et al., '08)
  - SFVS: $O(1.76^n)$ (Fomin et al., '16), $O(1.86^n)$ (weighted, Fomin et al., '13)
  - SFVS: chordal $O(1.68^n)$ (Golovach, '14), AT-free $O(1.62^n)$ (Chitnis, '17)
- Restricted to graph classes
  - FVS NP-complete: bipartite and planar
  - FVS $\in \mathcal{P}$: chordal (Spinrad, 2003), AT-free (Kratsch et al., 2008)
  - SFVS NP-complete: split (Fomin et al., 2013)
  - SFVS $\in \mathcal{P}$: **?**



Our Results

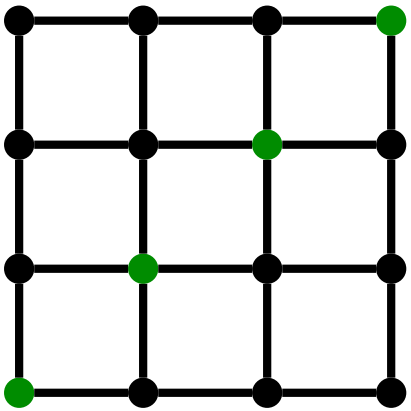
- Both problems are NP-complete (Garey and Johnson, '79)
- Exact algorithms
  - FVS: $O(1.75^n)$ (Raman et al., '08), $O(1.86^n)$ (weighted, Fomin et al., '08)
  - SFVS: $O(1.76^n)$ (Fomin et al., '16), $O(1.86^n)$ (weighted, Fomin et al., '13)
  - SFVS: chordal $O(1.68^n)$ (Golovach, '14), AT-free $O(1.62^n)$ (Chitnis, '17)
- Restricted to graph classes
  - FVS NP-complete: bipartite and planar
  - FVS $\in \mathcal{P}$: chordal (Spinrad, 2003), AT-free (Kratsch et al., 2008)
  - SFVS NP-complete: split (Fomin et al., 2013)
  - SFVS $\in \mathcal{P}$: **co-bipartite**, **interval**, **permutation**



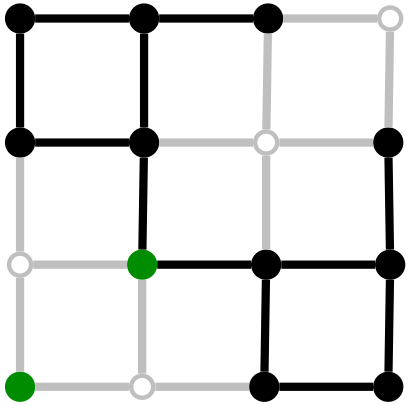
Maximal S-forests

- An SFVS X is *minimal* if no set $X' \subset X$ is an SFVS.

G



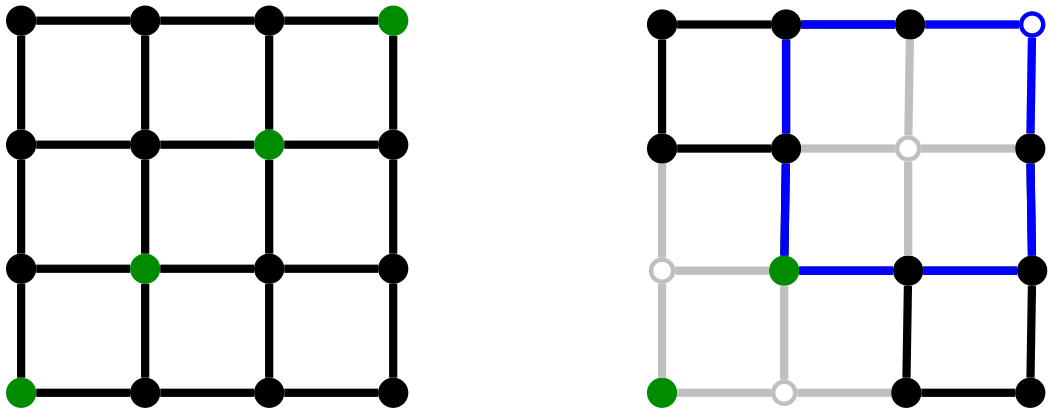
$G - X$



Maximal S-forests

- An SFVS X is *minimal* if no set $X' \subset X$ is an SFVS.

G $\qquad$ $G - X$



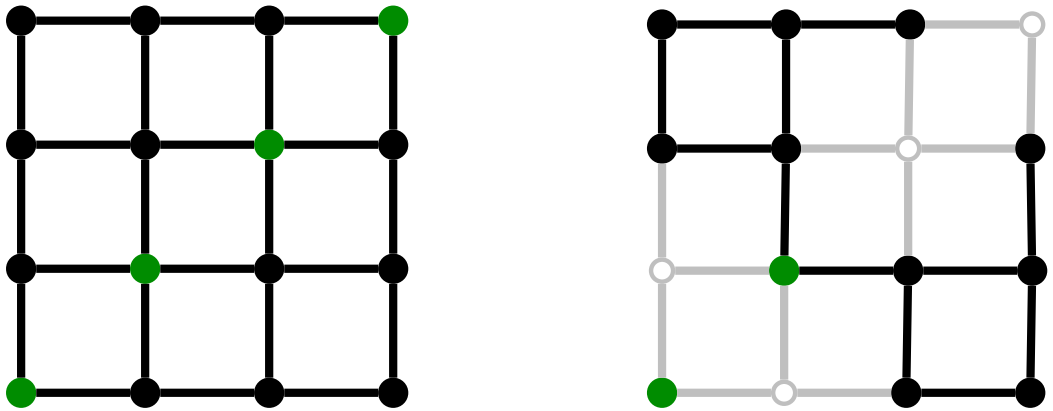
Every vertex v of a minimal X:

is the unique that is deleted from some S-cycle $\Rightarrow C_v$ (certifying cycle)

Maximal S-forests

- An SFVS X is *minimal* if no set $X' \subset X$ is an SFVS.

G $\qquad$ $G - X$



Every vertex v of a minimal X:

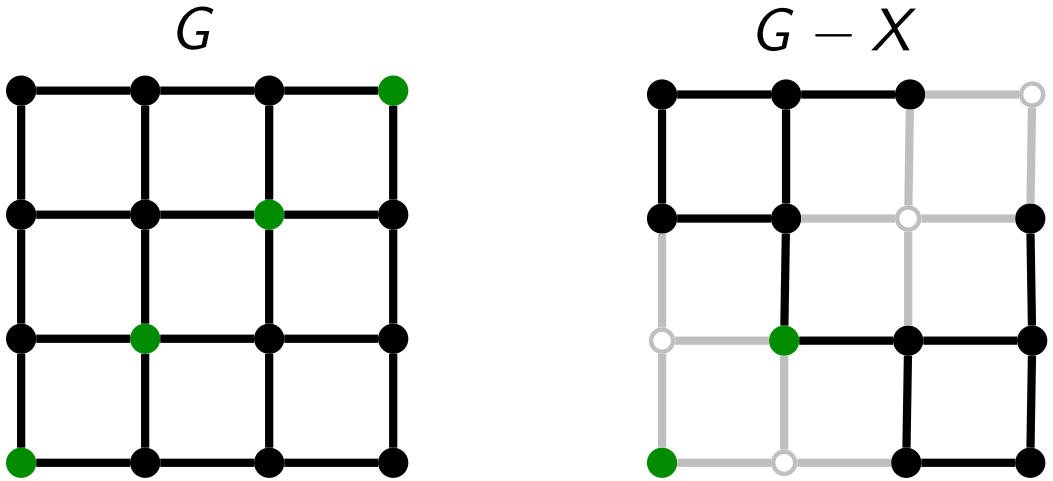
is the unique that is deleted from some S-cycle $\Rightarrow C_v$ (certifying cycle)

If X is a minimal SFVS:

- $Y = V - X$ is a **maximal S-forest**
- $\mathcal{F}_S$: the set of maximal S-forests

Maximal S-forests

- An SFVS X is *minimal* if no set $X' \subset X$ is an SFVS.



Every vertex v of a minimal X:

is the unique that is deleted from some S-cycle $\Rightarrow C_v$ (certifying cycle)

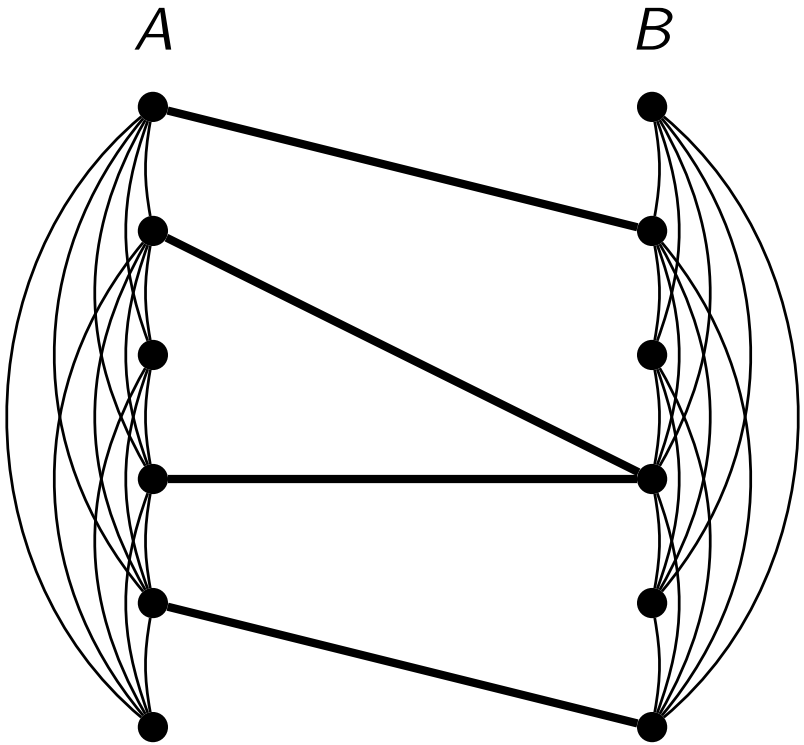
If X is a minimal SFVS:

- $Y = V - X$ is a **maximal S-forest**
- $\mathcal{F}_S$: the set of maximal S-forests
- The maximum solution is among $\mathcal{F}_S \Rightarrow$

Enumerate all maximal S-forests

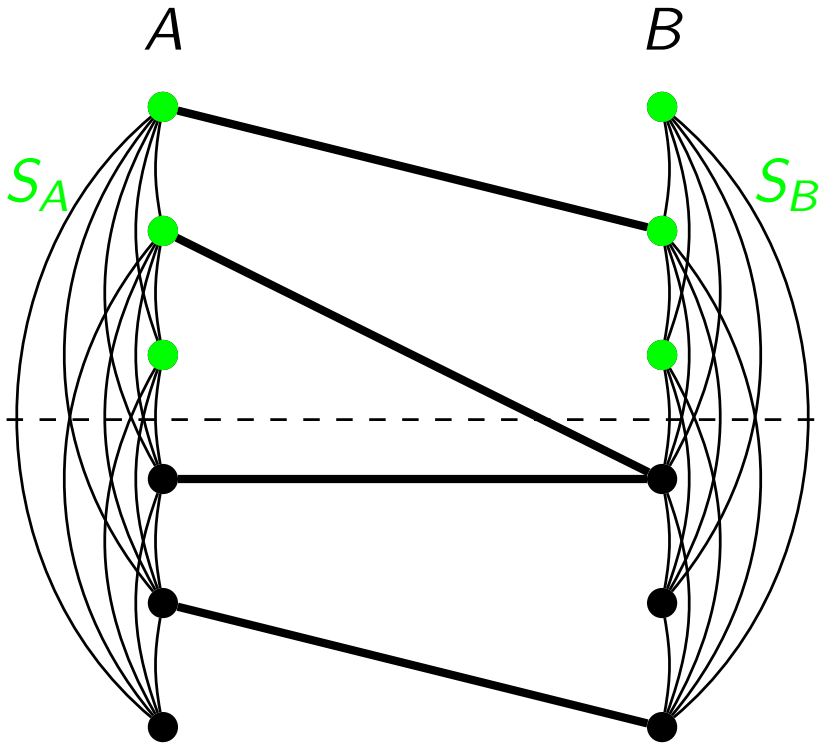
Complements of bipartite graphs

- co-bipartite graphs: V is partitioned into two cliques A and B



Complements of bipartite graphs

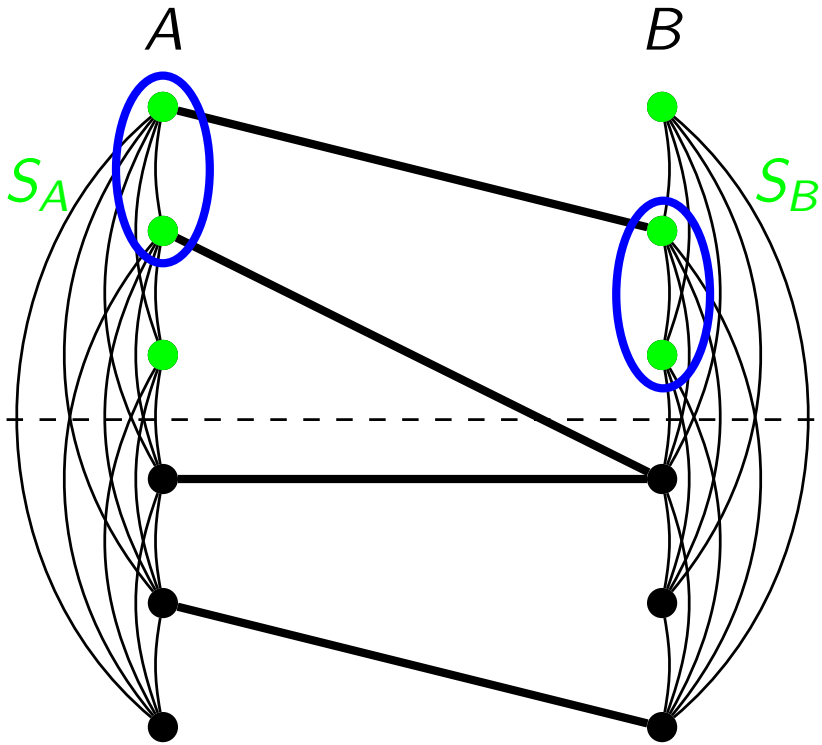
- co-bipartite graphs: V is partitioned into two cliques A and B



- $S_A = A \cap S$ and $S_B = B \cap S$

Complements of bipartite graphs

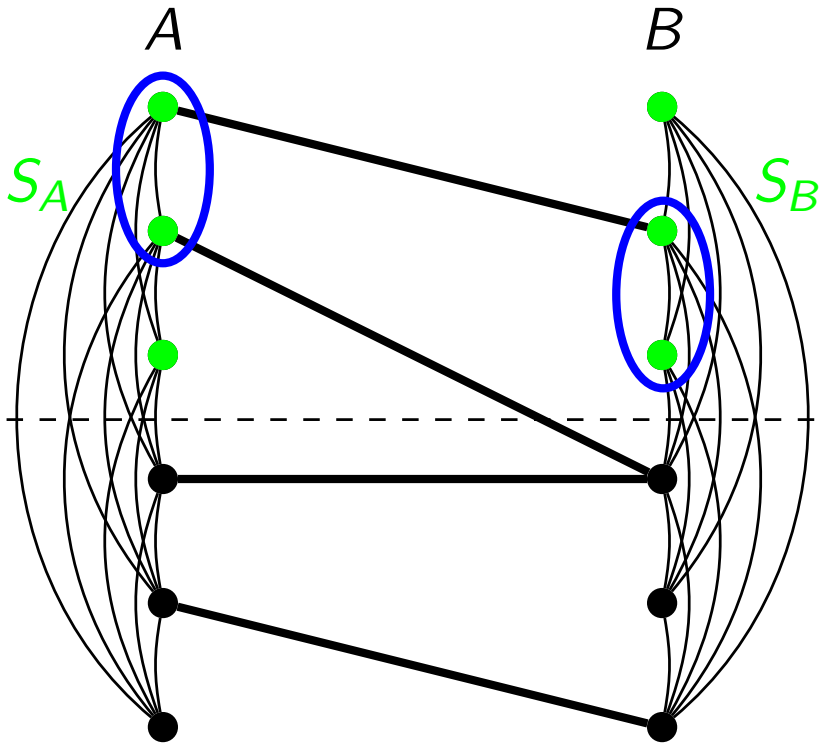
- co-bipartite graphs: V is partitioned into two cliques A and B



- $S_A = A \cap S$ and $S_B = B \cap S$
- For any maximal S-forest $F \Rightarrow |F \cap S_A| \leq 2$ and $|F \cap S_B| \leq 2$

Complements of bipartite graphs

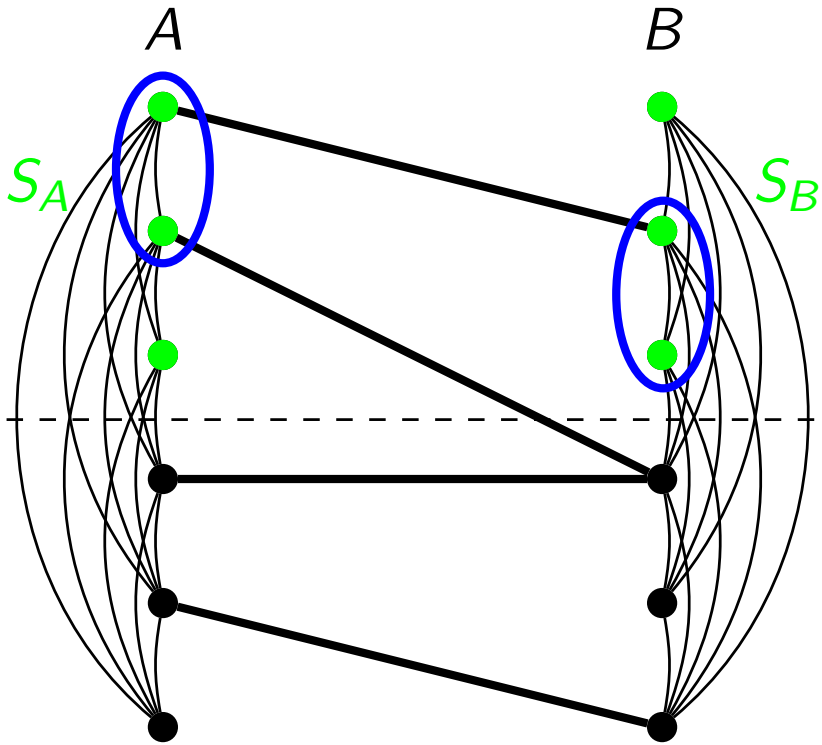
- co-bipartite graphs: V is partitioned into two cliques A and B



- $S_A = A \cap S$ and $S_B = B \cap S$
- For any maximal S-forest $F \Rightarrow |F \cap S_A| \leq 2$ and $|F \cap S_B| \leq 2$
- There are at most $22n^4$ maximal S-forests

Complements of bipartite graphs

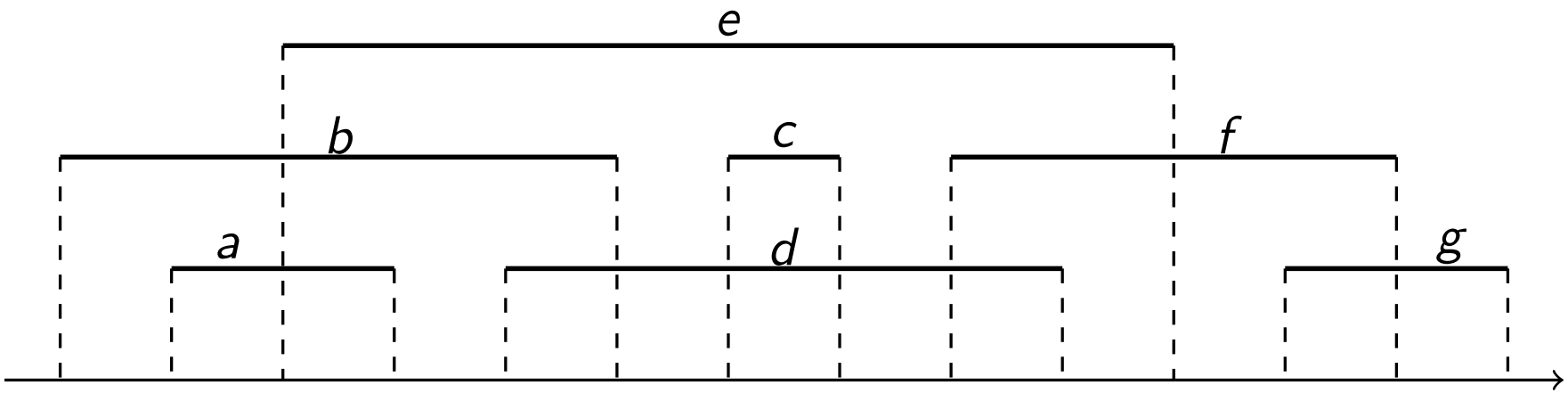
- co-bipartite graphs: V is partitioned into two cliques A and B



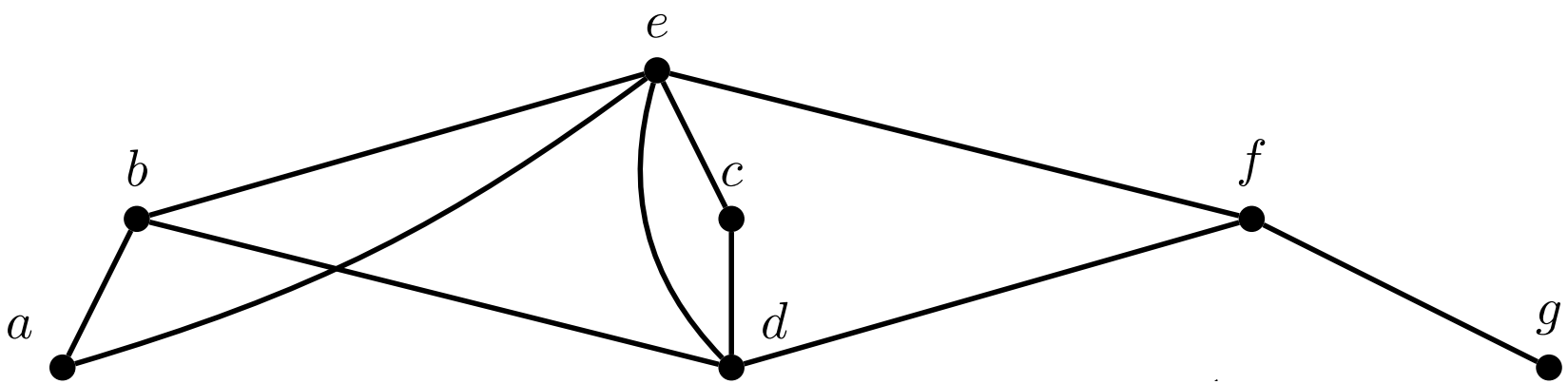
- $S_A = A \cap S$ and $S_B = B \cap S$
- For any maximal S-forest $F \Rightarrow |F \cap S_A| \leq 2$ and $|F \cap S_B| \leq 2$
- There are at most $22n^4$ maximal S-forests

$\Rightarrow$ An $O(n^4)$ algorithm for computing SFVS on co-biparite graphs.

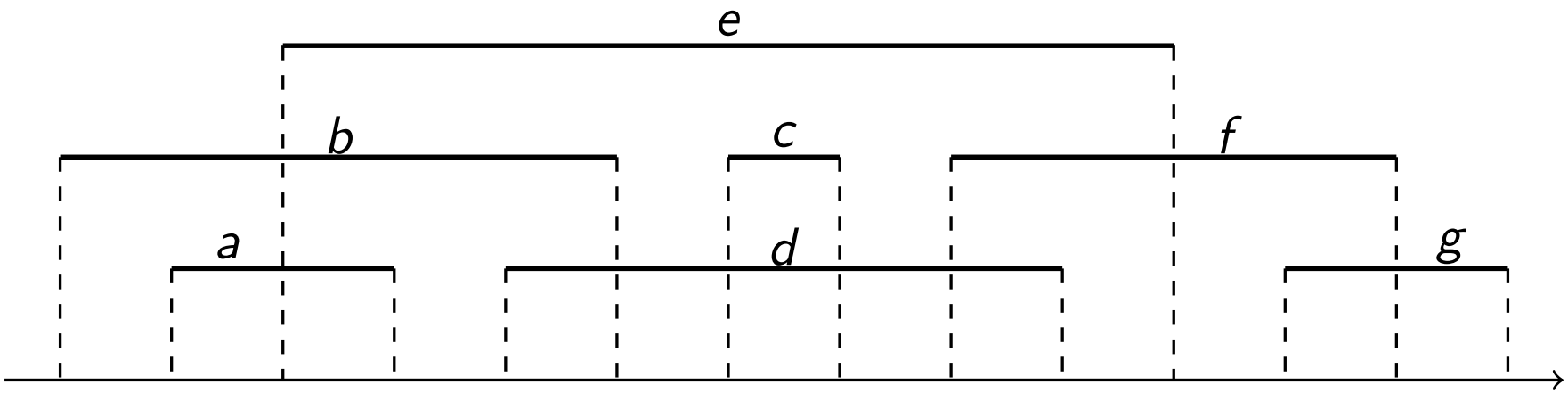
Interval Graphs



- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle

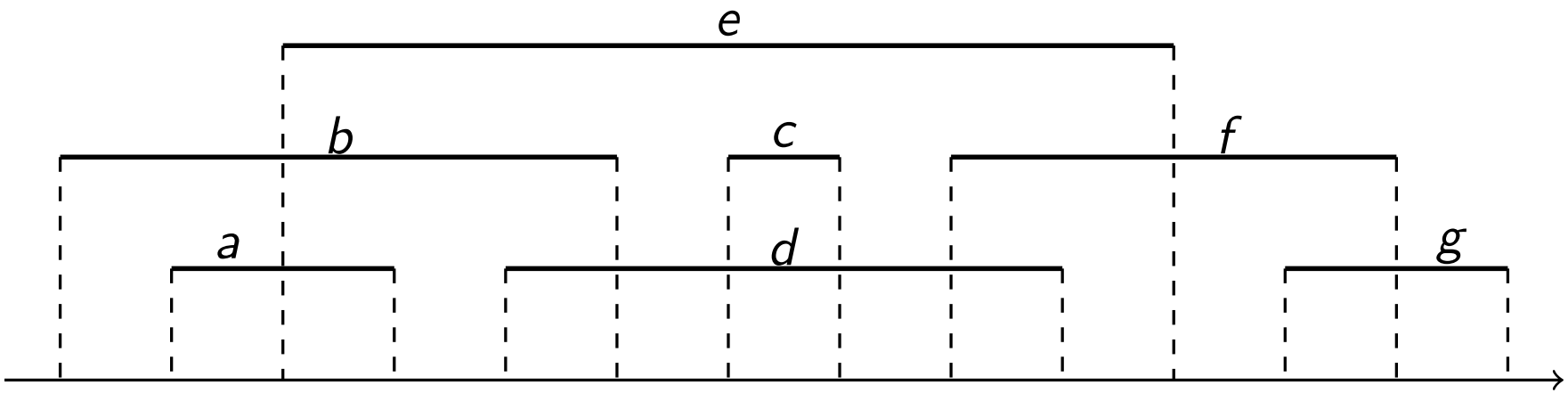


Interval Graphs



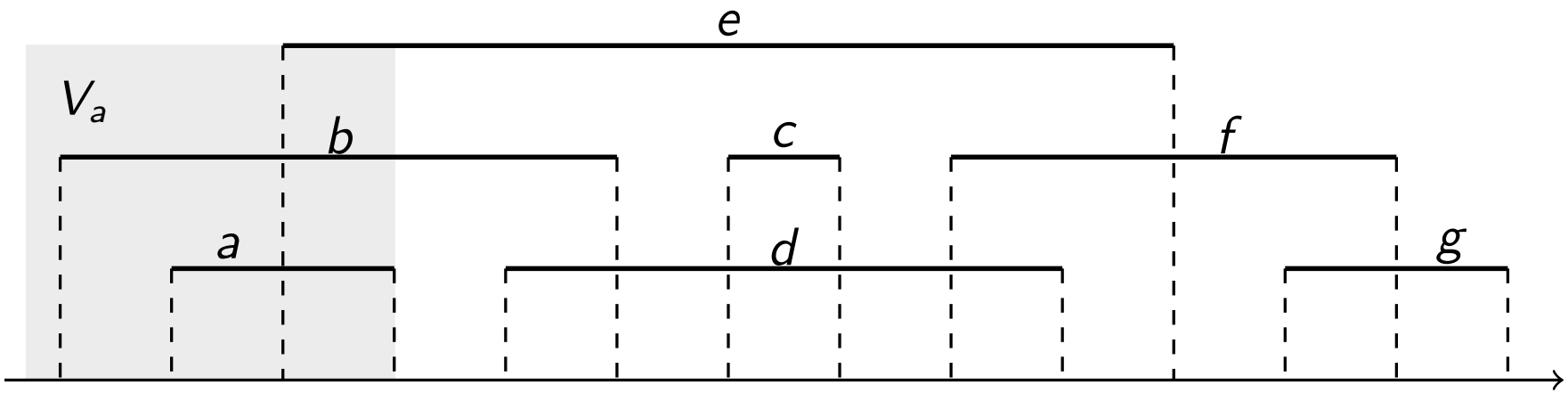
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach

Interval Graphs



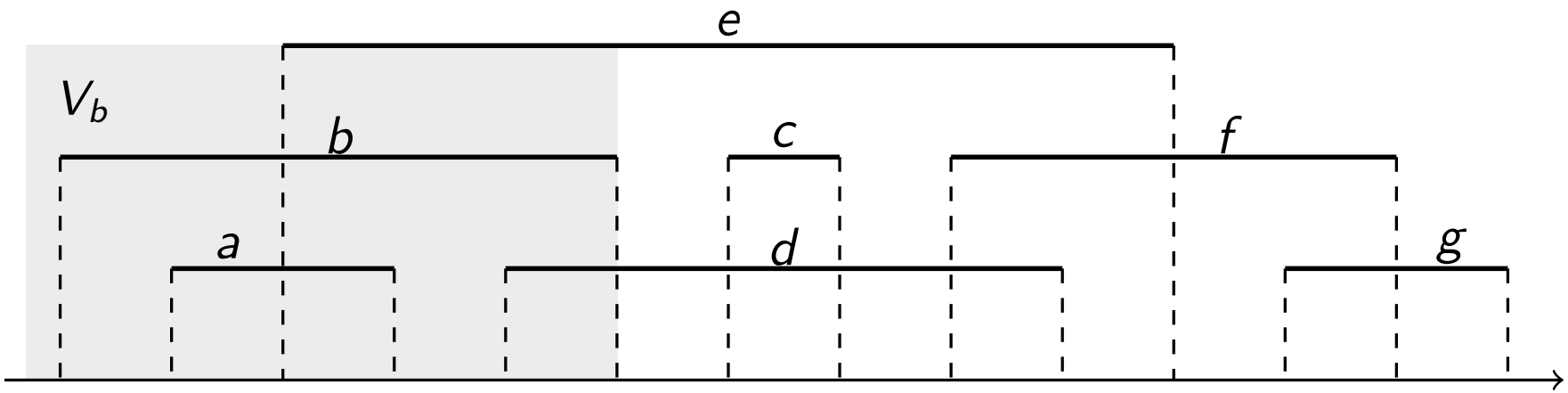
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint

Interval Graphs



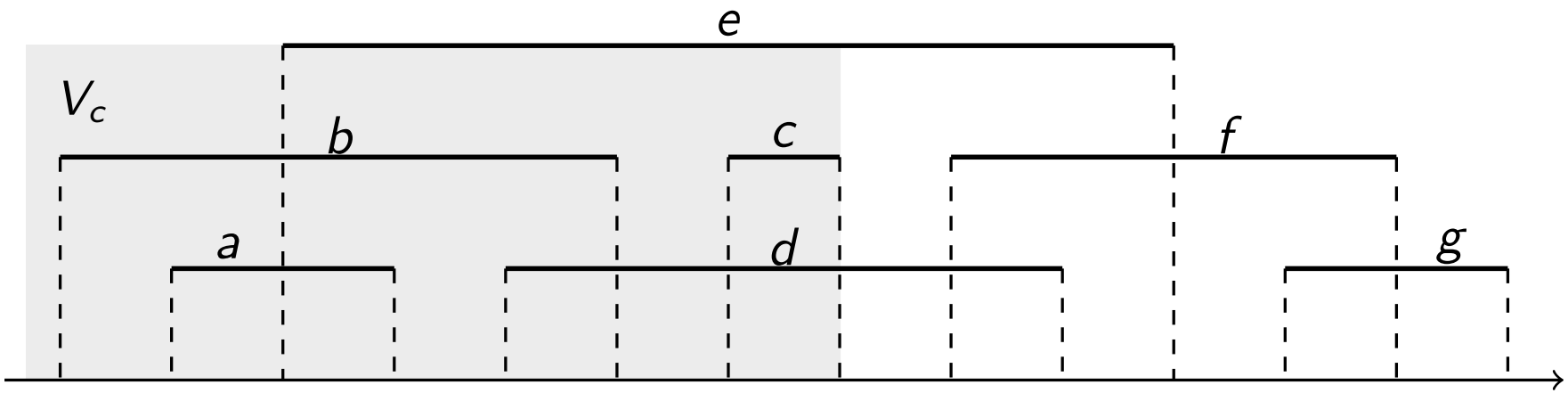
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Interval Graphs



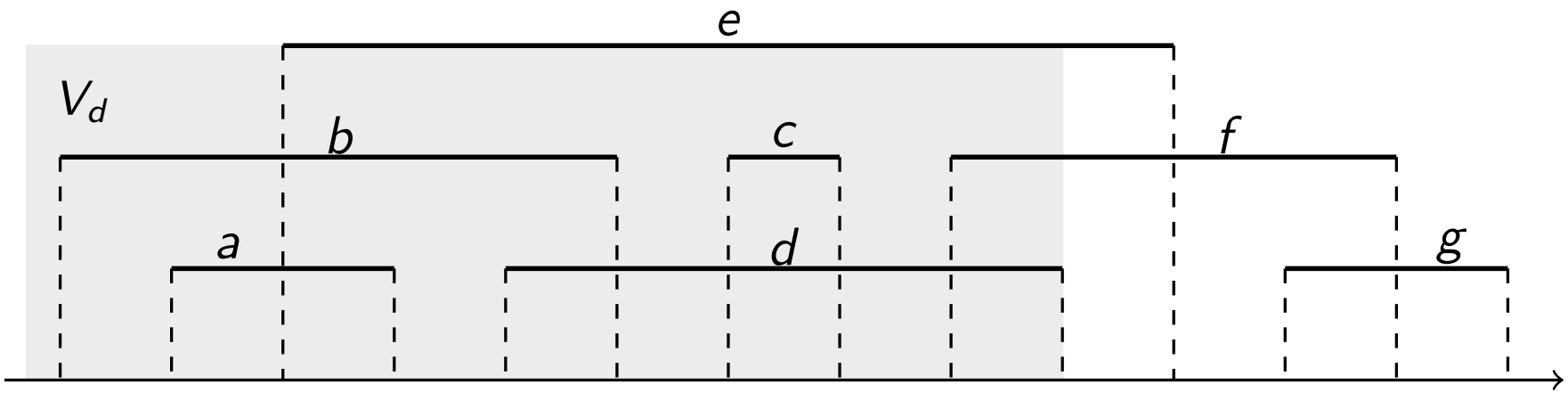
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Interval Graphs



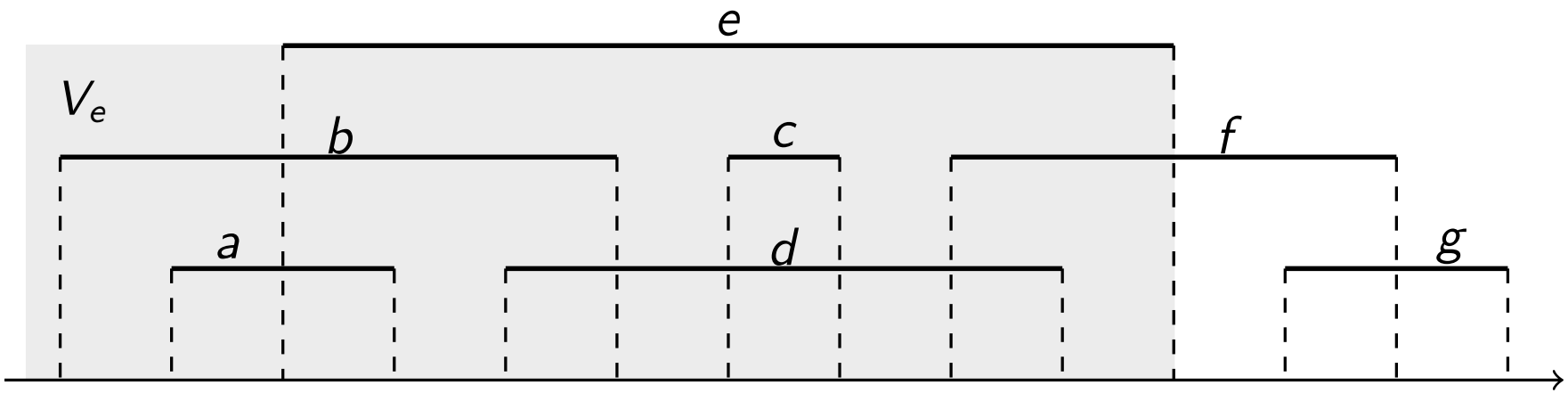
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Interval Graphs



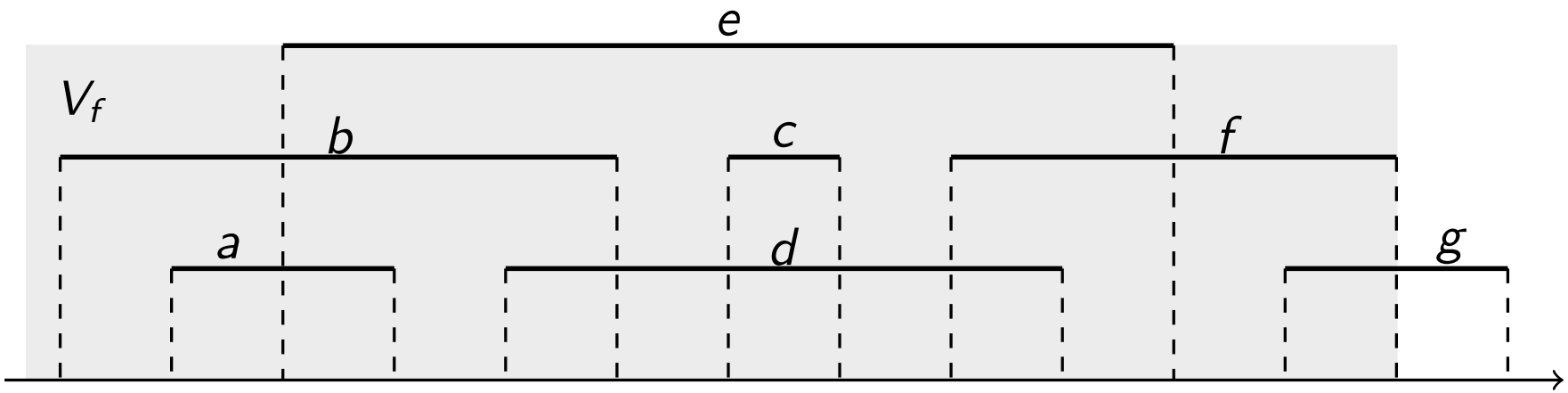
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Interval Graphs



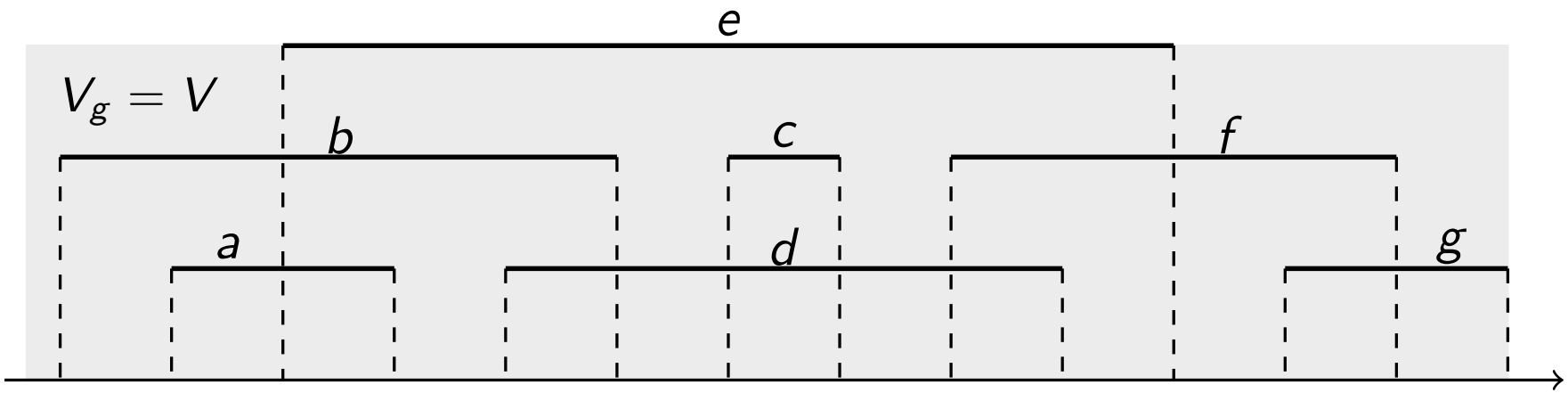
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Interval Graphs



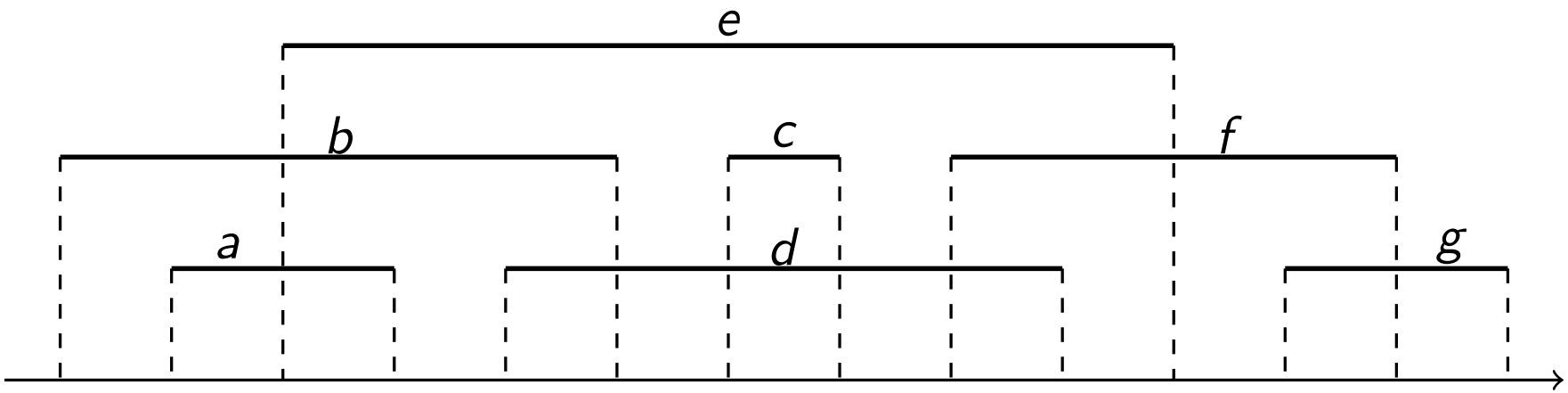
- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Interval Graphs

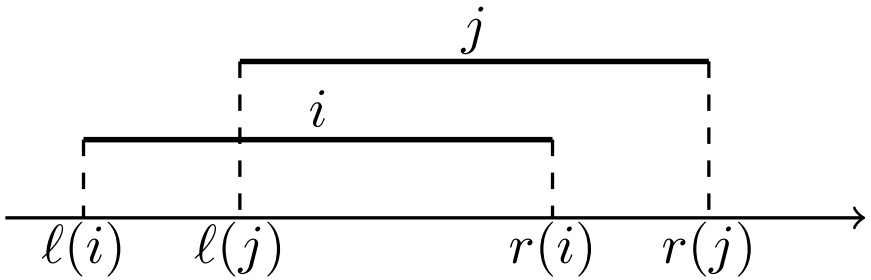


- $\mathcal{I}$: representation of closed intervals
- interval graphs: $V \leftrightarrow \mathcal{I}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of an interval graph is a triangle
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{I}$
- Dynamic programming approach
- Scan vertices from left-to-right according to their right endpoint
- We grow appropriately each maximal S-forest

Predecessors

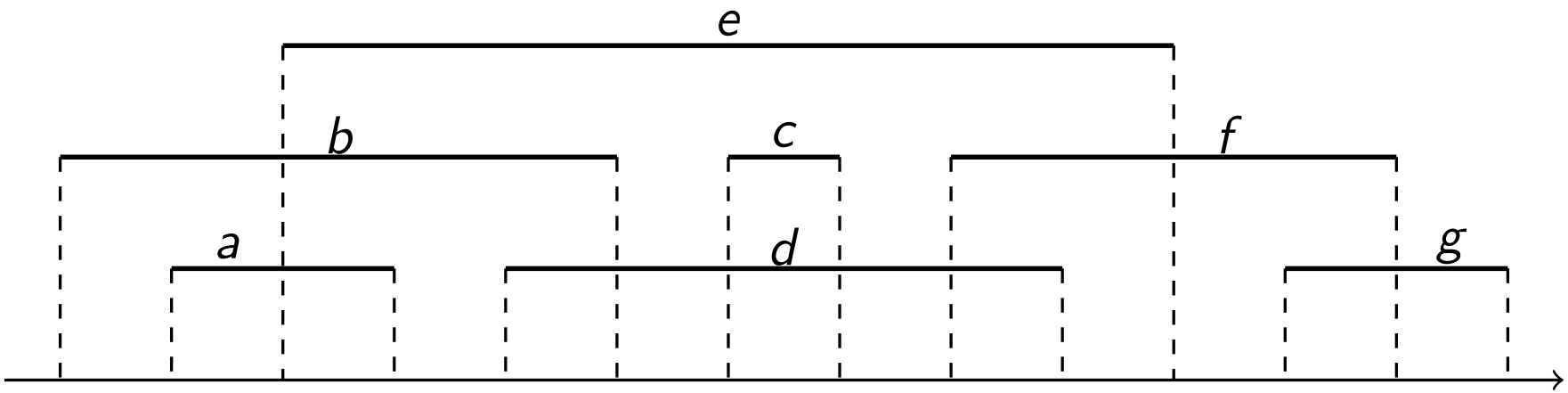


- The left and right endpoints of the intervals define $\leq_\ell$ and $\leq_r$:

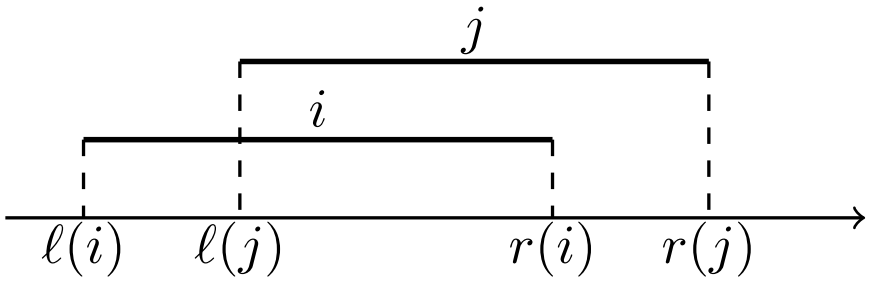


$$i \leq_l j \iff \ell(i) \leq \ell(j)$$
$$i \leq_r j \iff r(i) \leq r(j)$$

Predecessors



- The left and right endpoints of the intervals define $\leq_\ell$ and $\leq_r$:

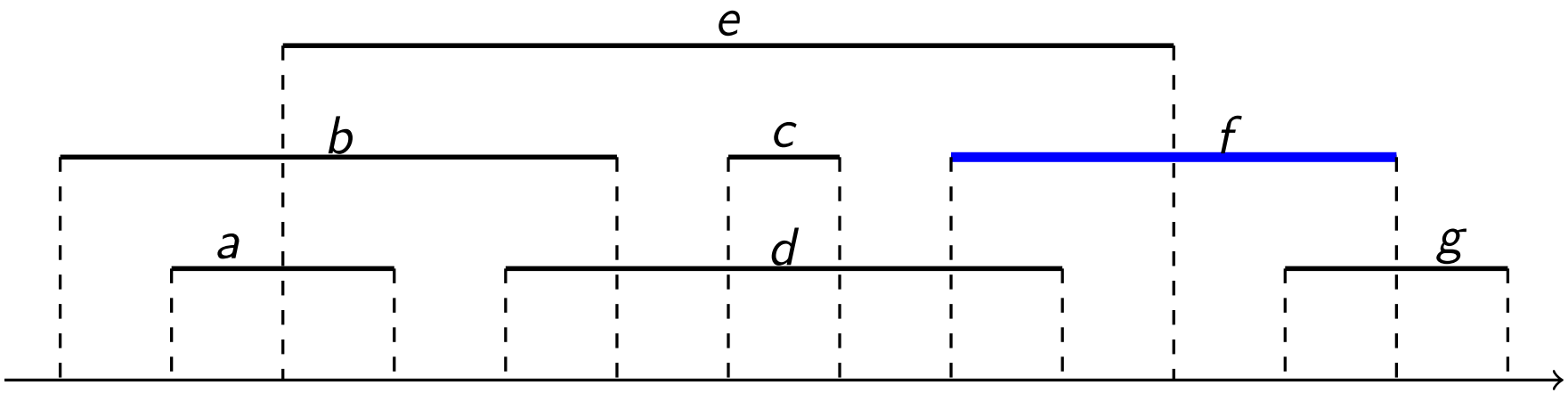


$$i \leq_l j \iff \ell(i) \leq \ell(j)$$
$$i \leq_r j \iff r(i) \leq r(j)$$

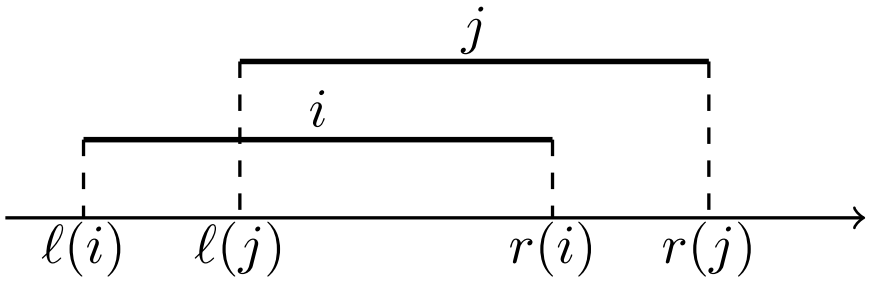
Predecessors of an interval i:

- $<i = r\text{-}\max(V_i \setminus \{i\})$
  the last interval that finishes before i finishes
- $\ll i = r\text{-}\max(V_i \setminus (\{i\} \cup \{h \in V : \{h, i\} \in E\}))$
  the last interval that finishes before i starts

Predecessors



- The left and right endpoints of the intervals define $\leq_\ell$ and $\leq_r$:

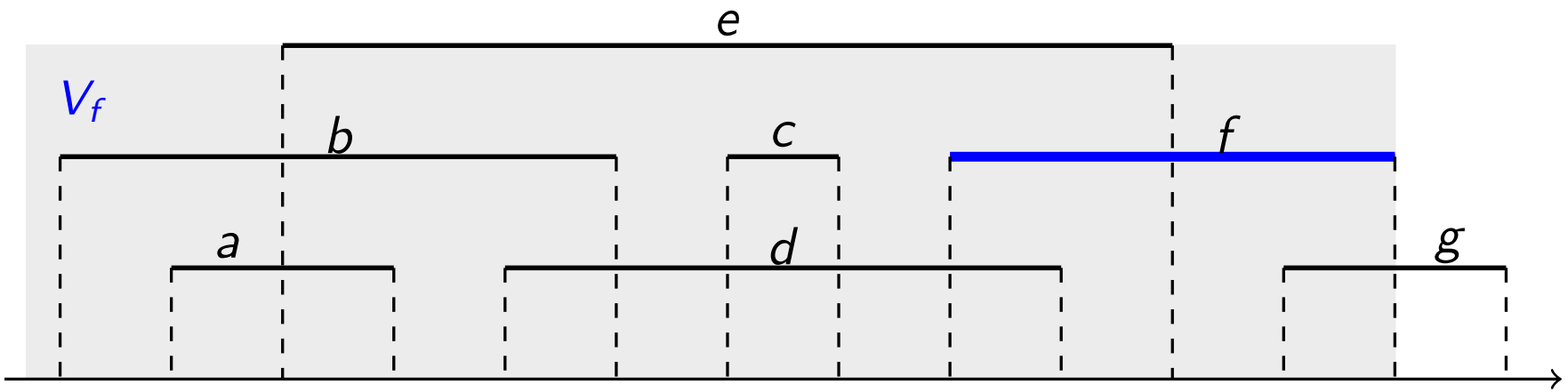


$$i \leq_l j \iff \ell(i) \leq \ell(j)$$
$$i \leq_r j \iff r(i) \leq r(j)$$

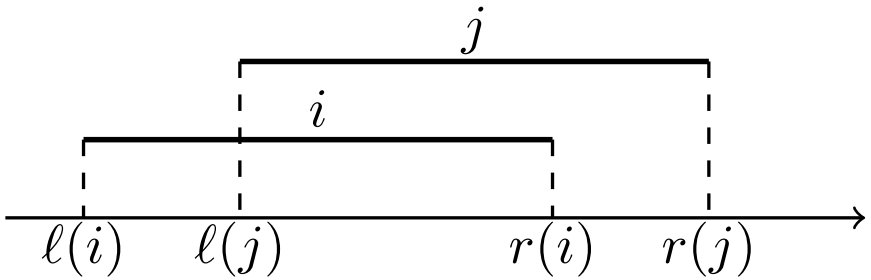
Predecessors of an interval i:

- $<i = r\text{-}\max(V_i \setminus \{i\})$
  the last interval that finishes before i finishes
- $\ll i = r\text{-}\max(V_i \setminus (\{i\} \cup \{h \in V : \{h, i\} \in E\}))$
  the last interval that finishes before i starts

Predecessors



- The left and right endpoints of the intervals define $\leq_\ell$ and $\leq_r$:

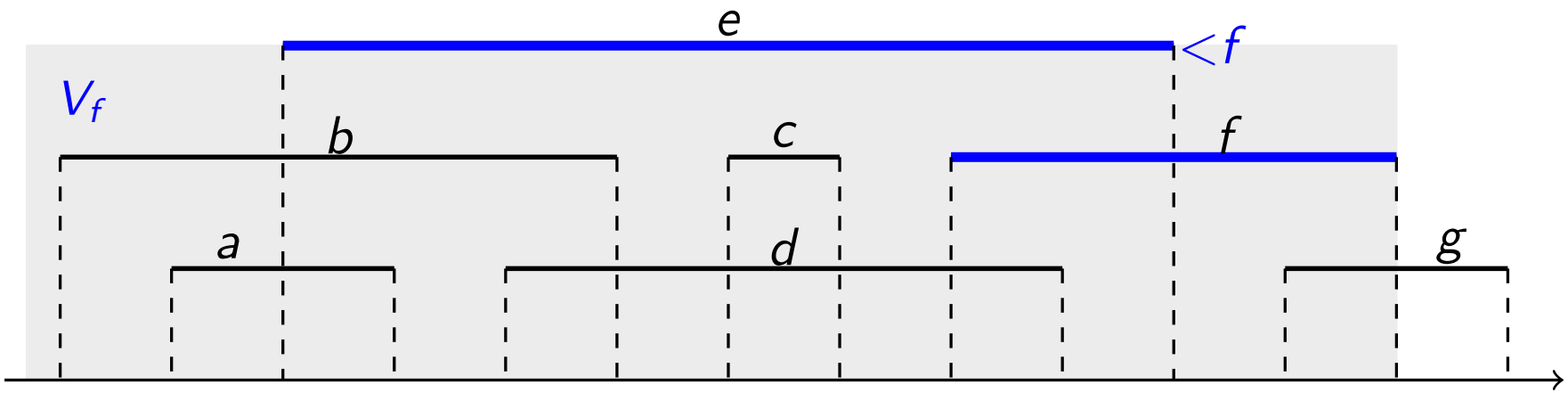


$$i \leq_l j \iff \ell(i) \leq \ell(j)$$
$$i \leq_r j \iff r(i) \leq r(j)$$

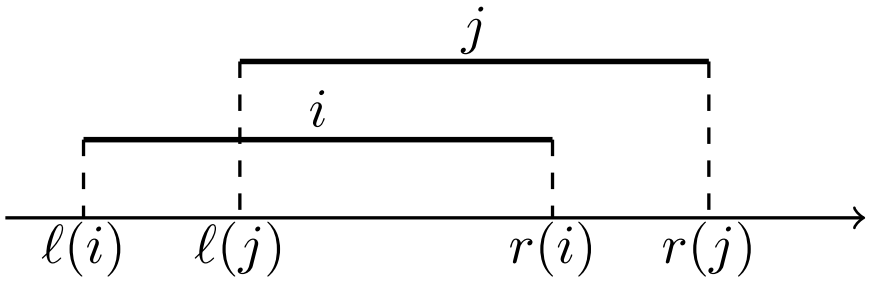
Predecessors of an interval i:

- $<i = r\text{-}\max(V_i \setminus \{i\})$
  the last interval that finishes before i finishes
- $\ll i = r\text{-}\max(V_i \setminus (\{i\} \cup \{h \in V : \{h, i\} \in E\}))$
  the last interval that finishes before i starts

Predecessors



- The left and right endpoints of the intervals define $\leq_\ell$ and $\leq_r$:

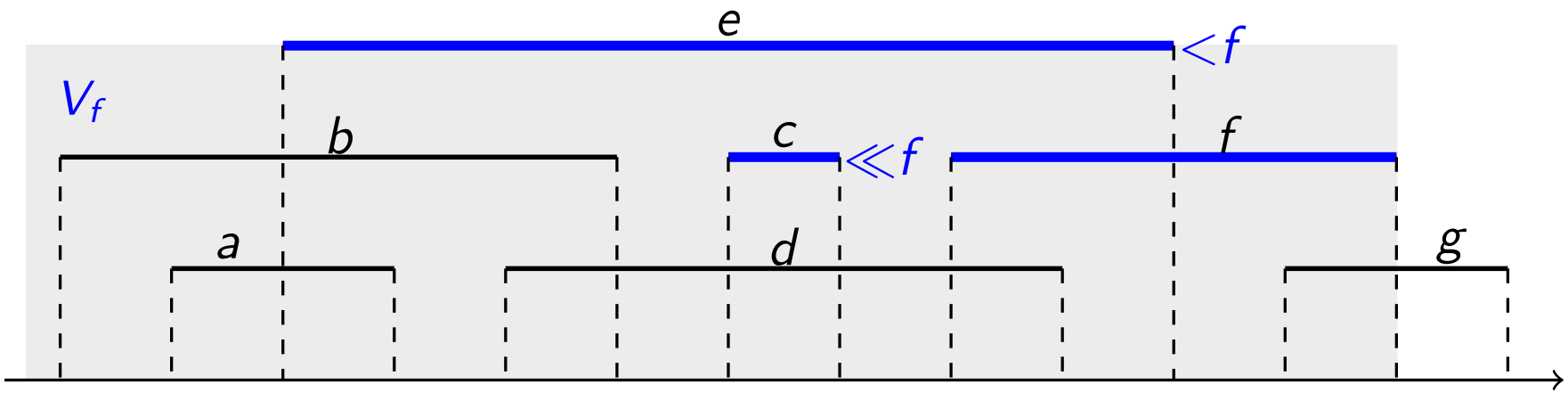


$$i \leq_l j \iff \ell(i) \leq \ell(j)$$
$$i \leq_r j \iff r(i) \leq r(j)$$

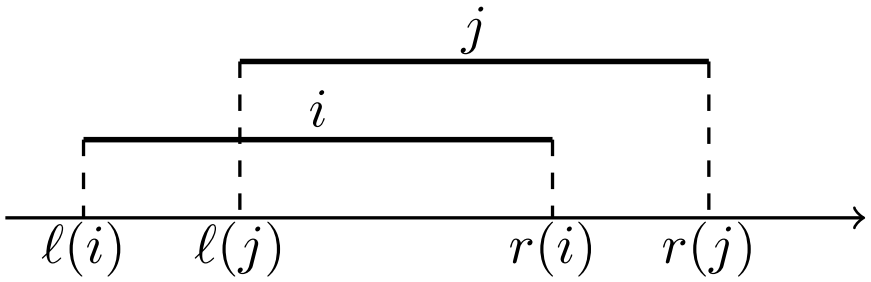
Predecessors of an interval i:

- $<i = r\text{-}\max(V_i \setminus \{i\})$
  the last interval that finishes before i finishes
- $\ll i = r\text{-}\max(V_i \setminus (\{i\} \cup \{h \in V : \{h, i\} \in E\}))$
  the last interval that finishes before i starts

Predecessors



- The left and right endpoints of the intervals define $\leq_\ell$ and $\leq_r$:



$$i \leq_l j \iff \ell(i) \leq \ell(j)$$
$$i \leq_r j \iff r(i) \leq r(j)$$

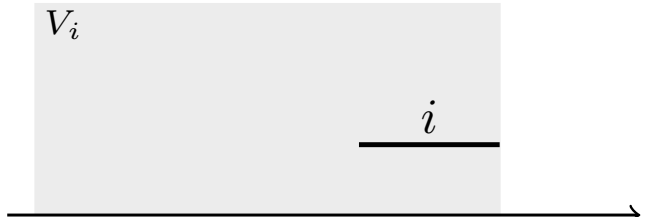
Predecessors of an interval i:

- $<i = r\text{-}\max(V_i \setminus \{i\})$
  the last interval that finishes before i finishes
- $\ll i = r\text{-}\max(V_i \setminus (\{i\} \cup \{h \in V : \{h, i\} \in E\}))$
  the last interval that finishes before i starts

Basic sets: A, B, C

- Sets used by our dynamic programming algorithm
- A: corresponds to a maximum S-forest
- B and C: choose a solution only from a predescribed set
- $x, y \in V - V_i$ such that $x <_\ell y$

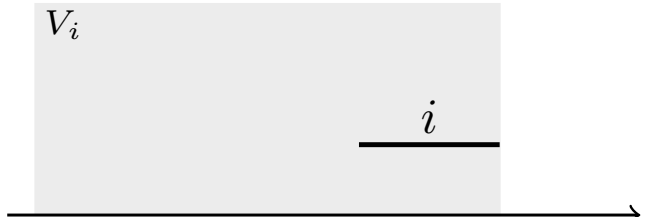
$$A_i = \max_w \{X \subseteq V_i : G[X] \in \mathcal{F}_S\}$$



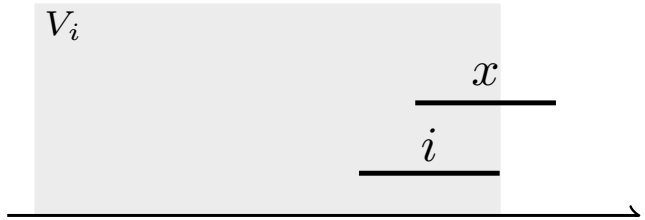
Basic sets: A, B, C

- Sets used by our dynamic programming algorithm
- A: corresponds to a maximum S-forest
- B and C: choose a solution only from a predescribed set
- $x, y \in V - V_i$ such that $x <_\ell y$

$$A_i = \max_w\{X \subseteq V_i : G[X] \in \mathcal{F}_S\}$$



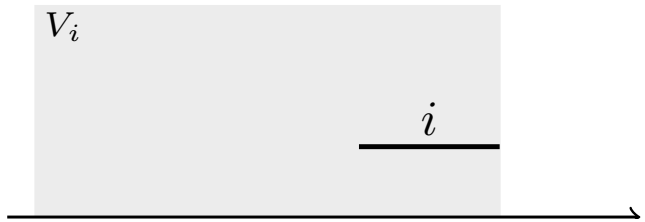
$$B_i^x = \max_w\{X \subseteq V_i \ : \ G[X \cup \{x\}] \in \mathcal{F}_S\}$$



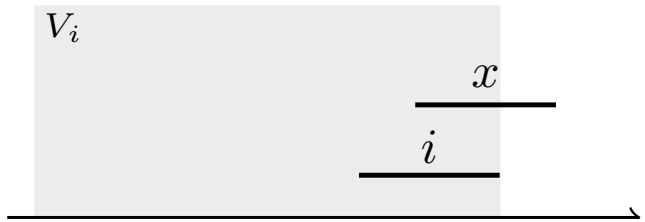
Basic sets: A, B, C

- Sets used by our dynamic programming algorithm
- A: corresponds to a maximum S-forest
- B and C: choose a solution only from a predescribed set
- $x, y \in V - V_i$ such that $x <_\ell y$

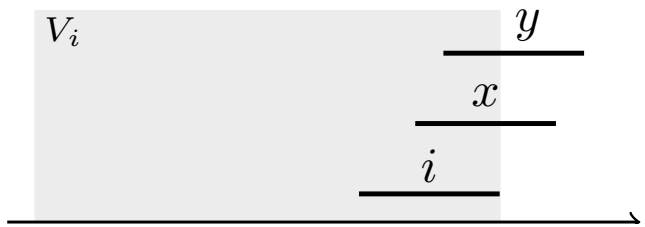
$$A_i = \max_w\{X \subseteq V_i : G[X] \in \mathcal{F}_S\}$$



$$B_i^x = \max_w\{X \subseteq V_i \ : \ G[X \cup \{x\}] \in \mathcal{F}_S\}$$



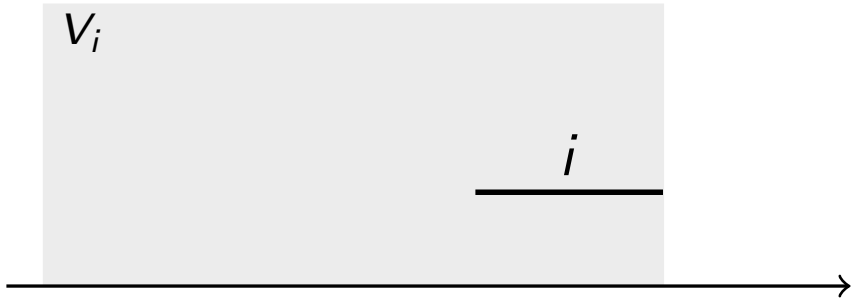
$$C_i^{x,y} = \max_w\{X \subseteq V_i : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$



Recursive formulation for A_i

A-set:

$$A_i = \max_w \left\{ A_{<i},\ B^i_{<i} \cup \{i\} \right\}$$



Recursive formulation for A_i

A-set:

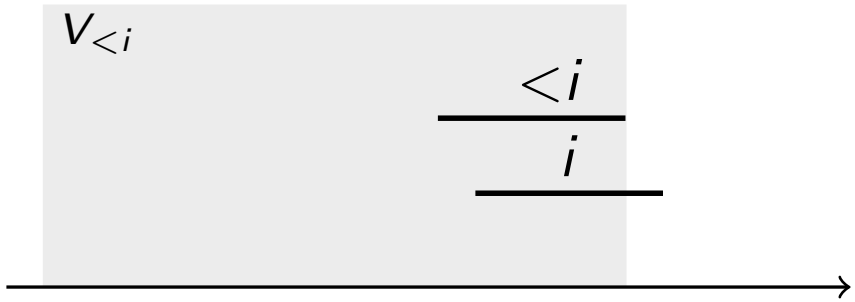
$$A_i = \max_w \left\{ A_{<i},\ B^i_{<i} \cup \{i\} \right\}$$

- If $i \notin A_i$ then i is irrelevant $\Rightarrow A_i = A_{<i}$

Recursive formulation for A_i

A-set:

$$A_i = \max_w \left\{ A_{<i},\ B^i_{<i} \cup \{i\} \right\}$$

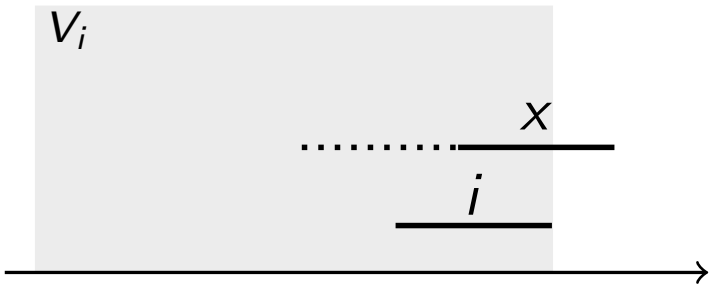


- If $i \notin A_i$ then i is irrelevant $\Rightarrow A_i = A_{<i}$
- If $i \in A_i$ then $B^i_{<i} \cup \{i\}$ contains no S-cycle

Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

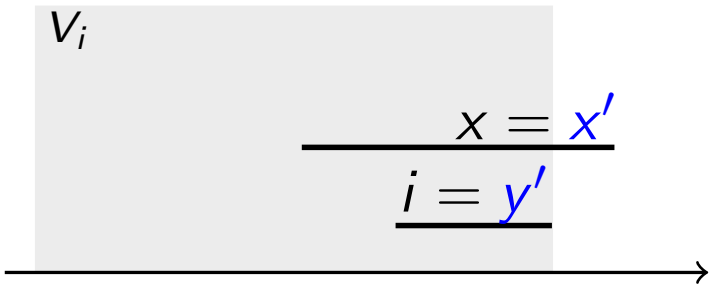
- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B_{<i}^x, B_{\ll y'}^{x'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B_{<i}^x, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$



Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

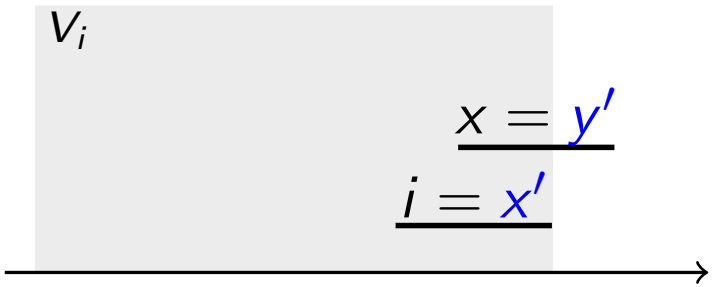
- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$



Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

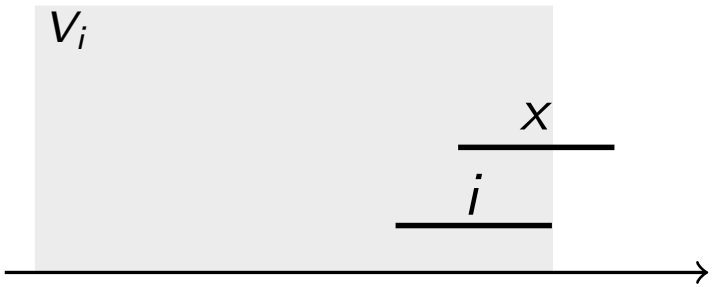
- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$



Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

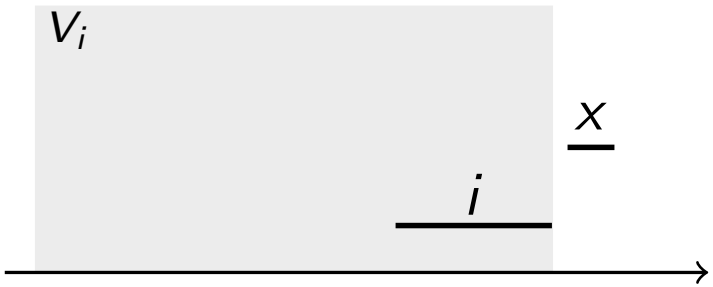
- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$



Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$

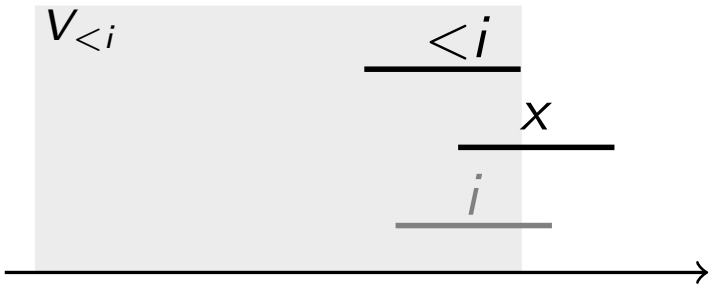


- If $\{i, x\} \notin E$: x is non-adjacent to any vertex of $V_i \Rightarrow A_i$

Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$

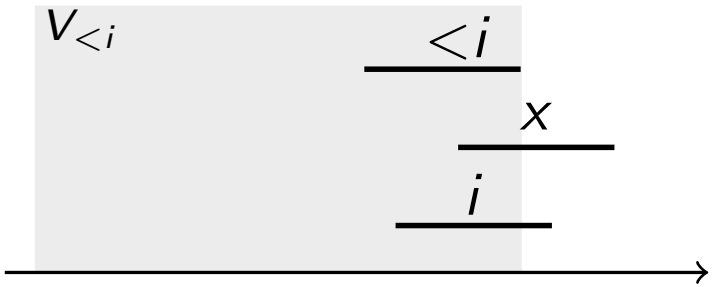


- If $\{i, x\} \notin E$: x is non-adjacent to any vertex of $V_i \Rightarrow A_i$
- If $\{i, x\} \in E$ and $i \notin B_i^x$: i is irrelevant $\Rightarrow B^x_{<i}$

Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$

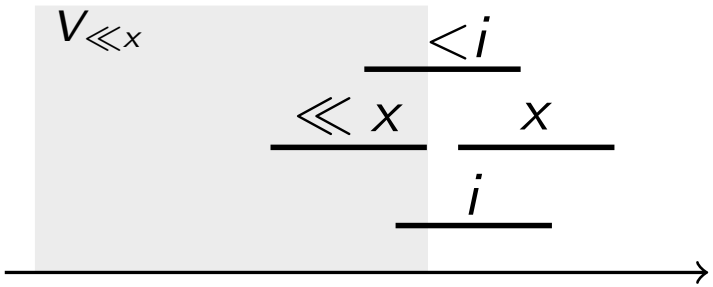


- If $\{i, x\} \notin E$: x is non-adjacent to any vertex of $V_i \Rightarrow A_i$
- If $\{i, x\} \in E$ and $i \notin B_i^x$: i is irrelevant $\Rightarrow B^x_{<i}$
- If $\{i, x\} \in E$ and $i \in B_i^x$:

Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$

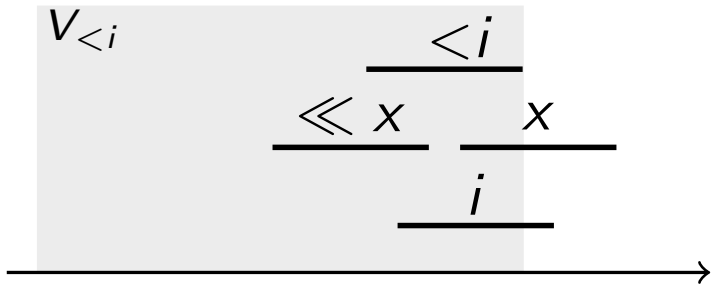


- If $\{i, x\} \notin E$: x is non-adjacent to any vertex of $V_i \Rightarrow A_i$
- If $\{i, x\} \in E$ and $i \notin B_i^x$: i is irrelevant $\Rightarrow B^x_{<i}$
- If $\{i, x\} \in E$ and $i \in B_i^x$:
  - i or $x \in S$: every vertex between $\ll x$ and i induces an S-triangle ($B^i_{\ll x}$)

Recursive formulation for B_i^x

Let $x' = \ell\text{-}\min\{i, x\}$ and let $y' = \{i, x\} \setminus x'$:

- If $\{i, x\} \notin E$, then $B_i^x = A_i$
- If $\{i, x\} \in E$, then $B_i^x = \begin{cases} \max_w \left\{ B^x_{<i}, B^{x'}_{\ll y'} \cup \{i\} \right\}, & \text{if } i \text{ or } x \in S \\ \max_w \left\{ B^x_{<i}, C^{x',y'}_{<i} \cup \{i\} \right\}, & \text{if } i, x \notin S \end{cases}$

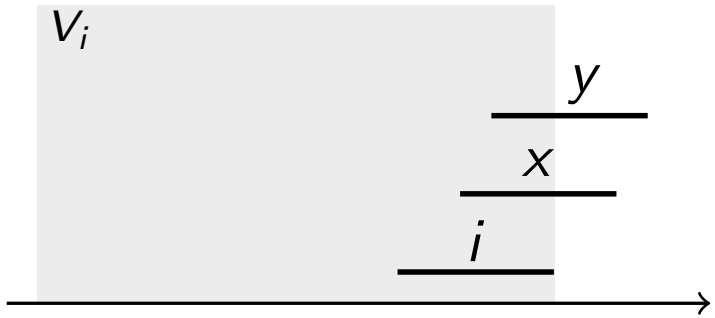


- If $\{i, x\} \notin E$: x is non-adjacent to any vertex of $V_i \Rightarrow A_i$
- If $\{i, x\} \in E$ and $i \notin B_i^x$: i is irrelevant $\Rightarrow B^x_{<i}$
- If $\{i, x\} \in E$ and $i \in B_i^x$:
  - i or $x \in S$: every vertex between $\ll x$ and i induces an S-triangle ($B^i_{\ll x}$)
  - $i \notin S$ and $x \notin S$: $C^{i,x}_{<i}$ contains no S-cycle

Recursive formulation for $C_i^{x,y}$

Let $x' = \ell\text{-}\min\{i, x, y\}$ and let $y' = \ell\text{-}\min(\{i, x, y\} \setminus \{x'\})$:

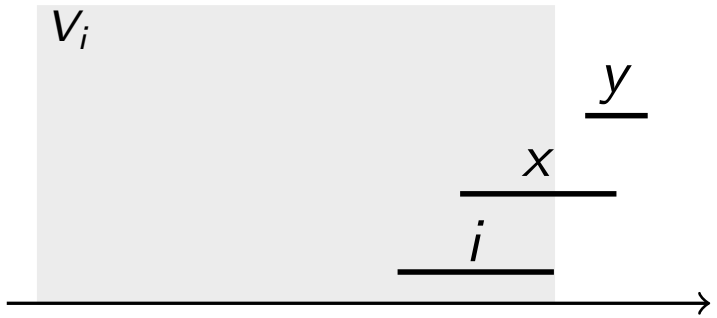
- If $\{i, y\} \notin E$, then $C_i^{x,y} = B_i^x$
- If $\{i, y\} \in E$, then $C_i^{x,y} = \begin{cases} C_{<i}^{x,y} & , \text{ if } i \in S \\ \max_w \left\{ C_{<i}^{x,y}, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i \notin S \end{cases}$



Recursive formulation for $C_i^{x,y}$

Let $x' = \ell\text{-}\min\{i, x, y\}$ and let $y' = \ell\text{-}\min(\{i, x, y\} \setminus \{x'\})$:

- If $\{i, y\} \notin E$, then $C_i^{x,y} = B_i^x$
- If $\{i, y\} \in E$, then $C_i^{x,y} = \begin{cases} C_{<i}^{x,y} & , \text{ if } i \in S \\ \max_w \left\{ C_{<i}^{x,y}, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i \notin S \end{cases}$

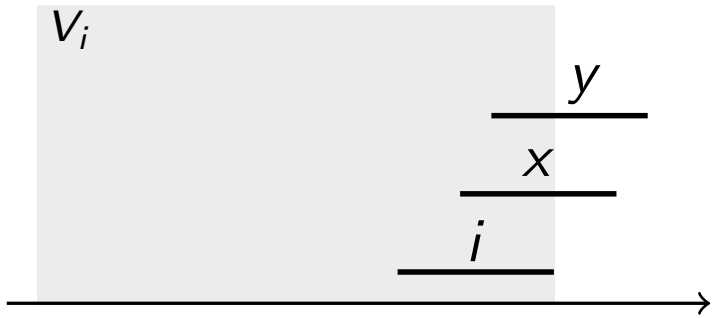


- If $\{i, y\} \notin E$: y is non-adjacent to any vertex of $V_i \Rightarrow B_i^x$

Recursive formulation for $C_i^{x,y}$

Let $x' = \ell\text{-}\min\{i, x, y\}$ and let $y' = \ell\text{-}\min(\{i, x, y\} \setminus \{x'\})$:

- If $\{i, y\} \notin E$, then $C_i^{x,y} = B_i^x$
- If $\{i, y\} \in E$, then $C_i^{x,y} = \begin{cases} C_{<i}^{x,y} & , \text{ if } i \in S \\ \max_w \left\{ C_{<i}^{x,y}, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i \notin S \end{cases}$

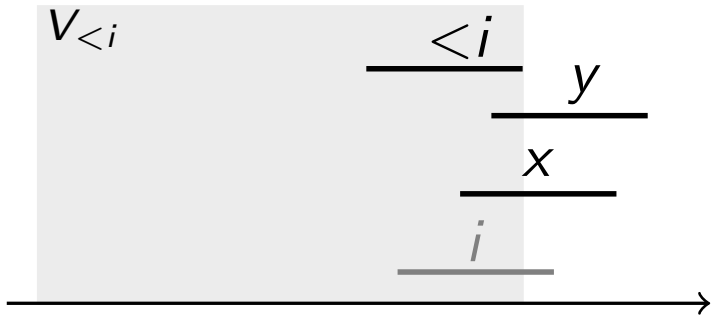


- If $\{i, y\} \notin E$: y is non-adjacent to any vertex of $V_i \Rightarrow B_i^x$
- If $\{i, y\} \in E$:
  - $i \in S$: $< i, x, y >$ is an S-triangle $\Rightarrow i \notin C_i^{x,y}$

Recursive formulation for $C_i^{x,y}$

Let $x' = \ell\text{-}\min\{i, x, y\}$ and let $y' = \ell\text{-}\min(\{i, x, y\} \setminus \{x'\})$:

- If $\{i, y\} \notin E$, then $C_i^{x,y} = B_i^x$
- If $\{i, y\} \in E$, then $C_i^{x,y} = \begin{cases} C_{<i}^{x,y} & , \text{ if } i \in S \\ \max_w \left\{ C_{<i}^{x,y}, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i \notin S \end{cases}$

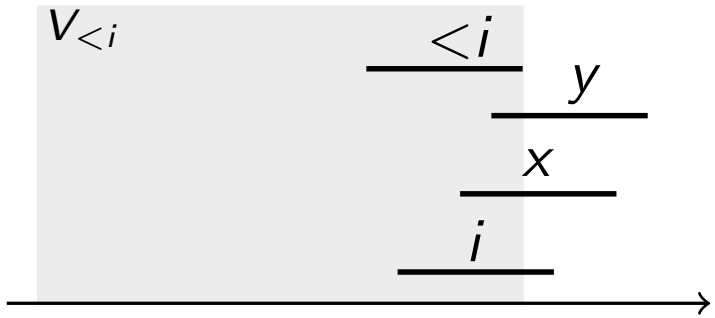


- If $\{i, y\} \notin E$: y is non-adjacent to any vertex of $V_i \Rightarrow B_i^x$
- If $\{i, y\} \in E$:
  - $i \in S$: $< i, x, y >$ is an S-triangle $\Rightarrow i \notin C_i^{x,y}$
  - $i \notin C_i^{x,y}$: i is irrelevant $\Rightarrow C_{<i}^{x,y}$

Recursive formulation for $C_i^{x,y}$

Let $x' = \ell\text{-}\min\{i, x, y\}$ and let $y' = \ell\text{-}\min(\{i, x, y\} \setminus \{x'\})$:

- If $\{i, y\} \notin E$, then $C_i^{x,y} = B_i^x$
- If $\{i, y\} \in E$, then $C_i^{x,y} = \begin{cases} C_{<i}^{x,y} & , \text{ if } i \in S \\ \max_w \left\{ C_{<i}^{x,y}, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i \notin S \end{cases}$

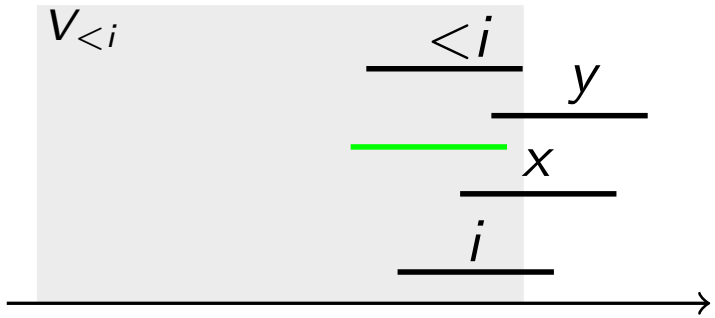


- If $\{i, y\} \notin E$: y is non-adjacent to any vertex of $V_i \Rightarrow B_i^x$
- If $\{i, y\} \in E$:
  - $i \in S$: $< i, x, y >$ is an S-triangle $\Rightarrow i \notin C_i^{x,y}$
  - $i \notin C_i^{x,y}$: i is irrelevant $\Rightarrow C_{<i}^{x,y}$
- If $\{i, y\} \in E$ and $i \in C_i^{x,y}$: $S \cap \{i, x, y\} = \emptyset$

Recursive formulation for $C_i^{x,y}$

Let $x' = \ell\text{-}\min\{i, x, y\}$ and let $y' = \ell\text{-}\min(\{i, x, y\} \setminus \{x'\})$:

- If $\{i, y\} \notin E$, then $C_i^{x,y} = B_i^x$
- If $\{i, y\} \in E$, then $C_i^{x,y} = \begin{cases} C_{<i}^{x,y} & , \text{ if } i \in S \\ \max_w \left\{ C_{<i}^{x,y}, C_{<i}^{x',y'} \cup \{i\} \right\}, & \text{if } i \notin S \end{cases}$



- If $\{i, y\} \notin E$: y is non-adjacent to any vertex of $V_i \Rightarrow B_i^x$
- If $\{i, y\} \in E$:
  - $i \in S$: $< i, x, y >$ is an S-triangle $\Rightarrow i \notin C_i^{x,y}$
  - $i \notin C_i^{x,y}$: i is irrelevant $\Rightarrow C_{<i}^{x,y}$
- If $\{i, y\} \in E$ and $i \in C_i^{x,y}$: $S \cap \{i, x, y\} = \emptyset$
  - $C_{<i}^{i,x} \cup \{i\}$: none of its subset can induce an S-cycle with y

Poly-time for SFVS on Interval Graphs

Theorem

There is a poly-time algorithm that computes SFVS of an interval graph.

Poly-time for SFVS on Interval Graphs

Theorem

There is a poly-time algorithm that computes SFVS of an interval graph.

1. We compute the predecessors $< i$ and $\ll i$ for each i in linear time

Poly-time for SFVS on Interval Graphs

Theorem

There is a poly-time algorithm that computes SFVS of an interval graph.

1. We compute the predecessors $< i$ and $\ll i$ for each i in linear time
2. Scan all intervals in an ascending order with respect to $<_\ell$:
   - Compute first A_i
   - Then compute B_i^x and $C_i^{x,y}$ for every x, y such that $\ell(i) < \ell(x) < \ell(y)$

Poly-time for SFVS on Interval Graphs

Theorem

There is a poly-time algorithm that computes SFVS of an interval graph.

1. We compute the predecessors $< i$ and $\ll i$ for each i in linear time
2. Scan all intervals in an ascending order with respect to $<_\ell$:
   - Compute first A_i
   - Then compute B_i^x and $C_i^{x,y}$ for every x, y such that $\ell(i) < \ell(x) < \ell(y)$
3. At the end, output A_n

Poly-time for SFVS on Interval Graphs

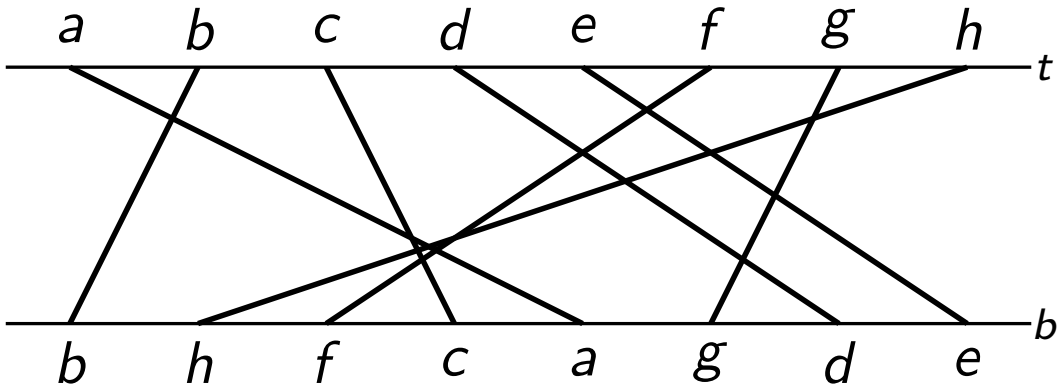
Theorem

There is a poly-time algorithm that computes SFVS of an interval graph.

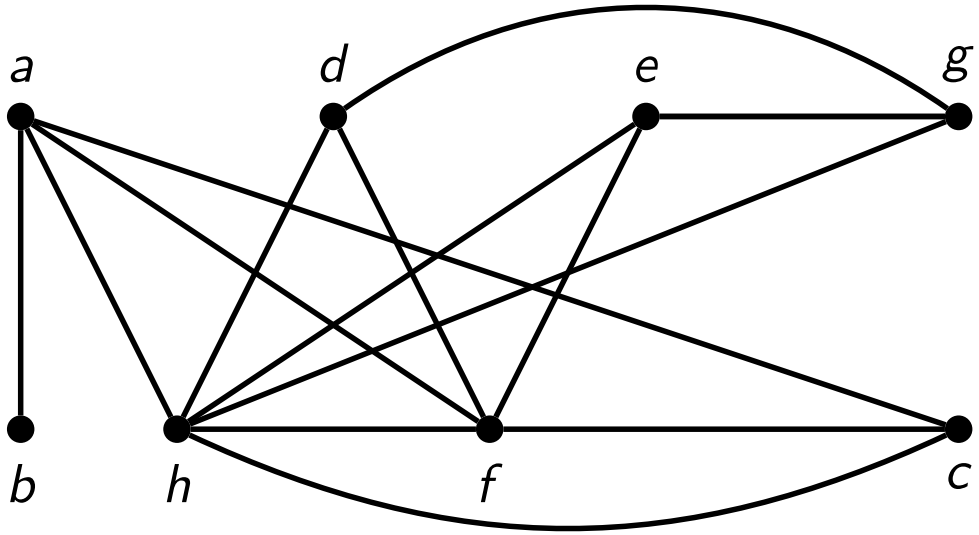
1. We compute the predecessors $< i$ and $\ll i$ for each i in linear time
2. Scan all intervals in an ascending order with respect to $<_\ell$:
   - Compute first A_i
   - Then compute B_i^x and $C_i^{x,y}$ for every x, y such that $\ell(i) < \ell(x) < \ell(y)$
3. At the end, output A_n

The total running time of the algorithm is $O(n^3)$.

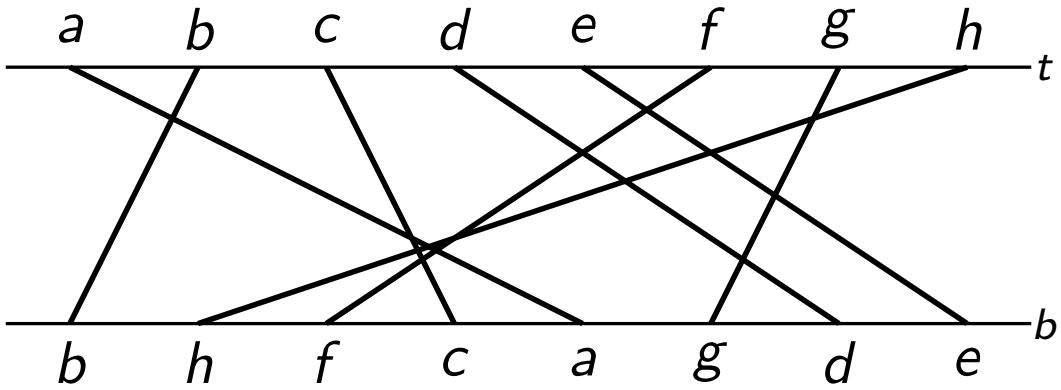
Permutation Graphs



- $\mathcal{D}$: permutation diagram with $\mathcal{S}$egments between two parallel lines
- permutation graphs: $V \leftrightarrow \mathcal{S}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of a permutation graph is a triangle or a square

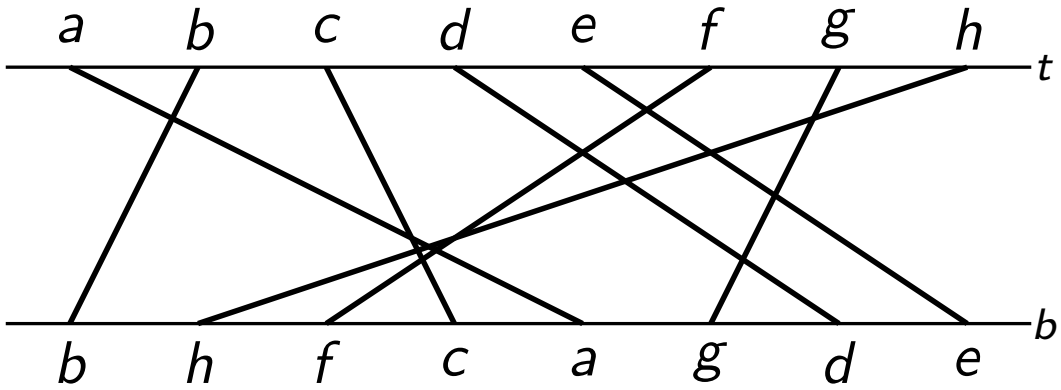


Permutation Graphs



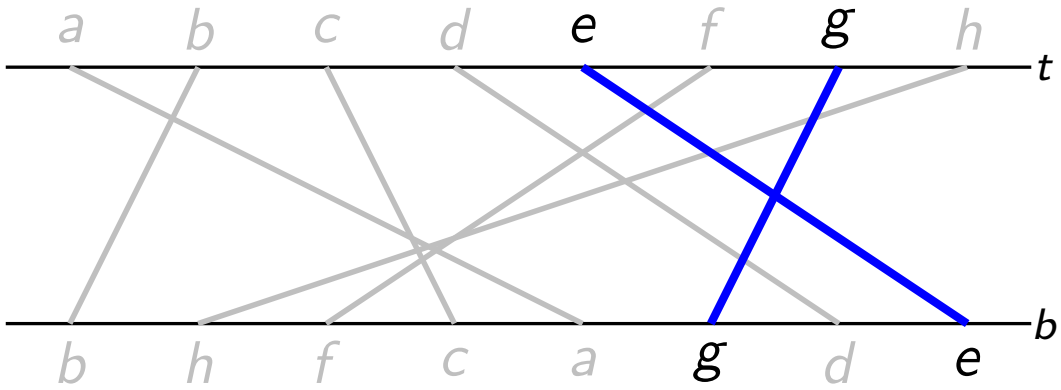
- $\mathcal{D}$: permutation diagram with $\mathcal{S}$egments between two parallel lines
- permutation graphs: $V \leftrightarrow \mathcal{S}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of a permutation graph is a triangle or a square
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{D}$
- Dynamic programming approach

Permutation Graphs



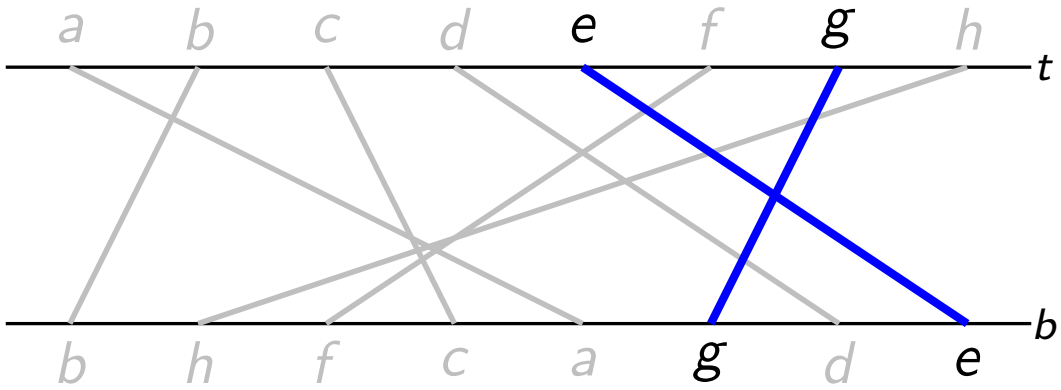
- $\mathcal{D}$: permutation diagram with $\mathcal{S}$egments between two parallel lines
- permutation graphs: $V \leftrightarrow \mathcal{S}$ such that $(u, v) \in E$ iff u and v intersect
- Every induced cycle of a permutation graph is a triangle or a square
- Compute maximal S-forests $\mathcal{F}_S$ based on $\mathcal{D}$
- Dynamic programming approach based on **crossing pairs**

Crossing Pairs



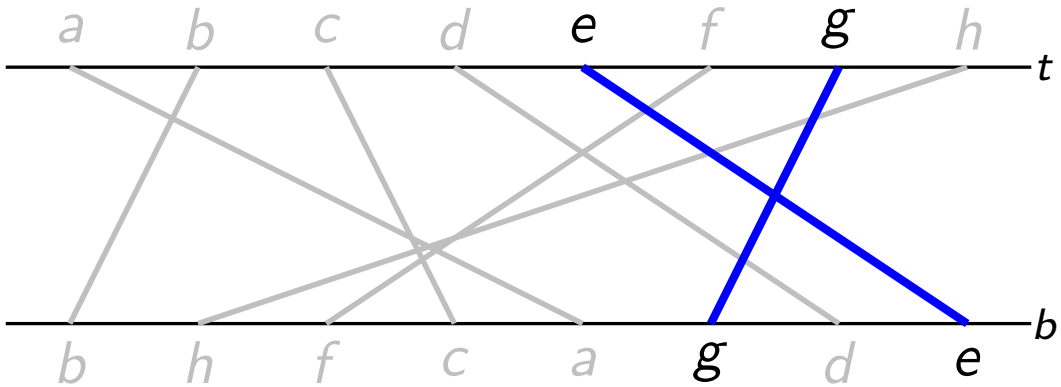
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$

Crossing Pairs



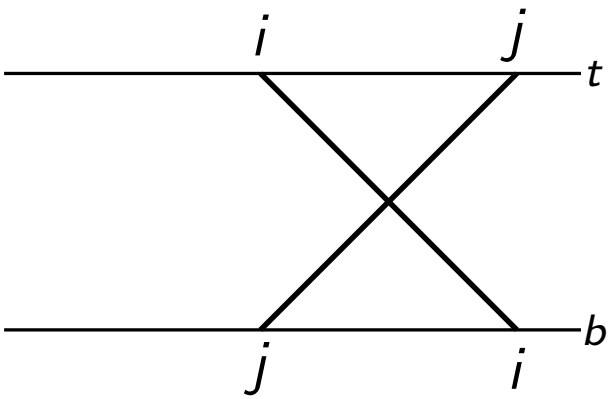
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

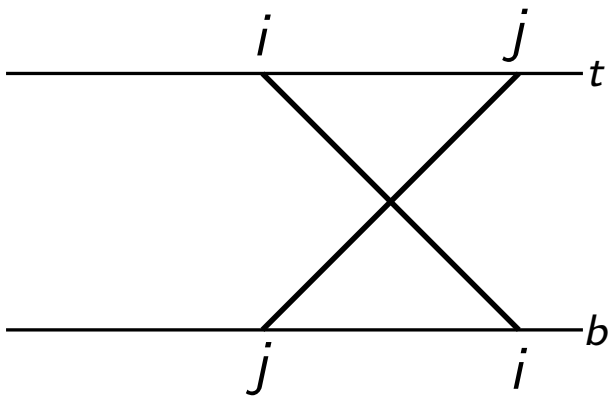
Crossing Pairs



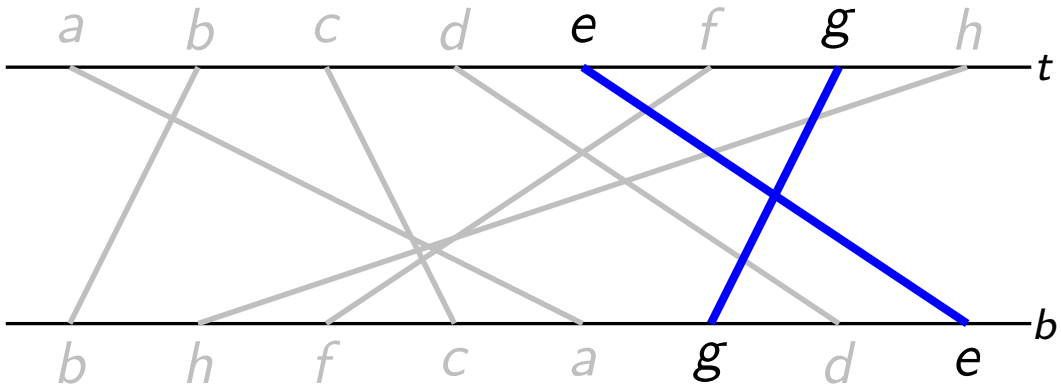
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

$$gh \leq_\ell ij \quad \Leftrightarrow \quad g \leq_t i \text{ and } h \leq_b j$$



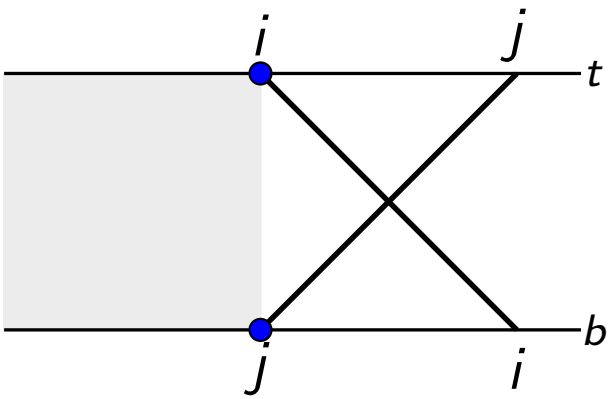


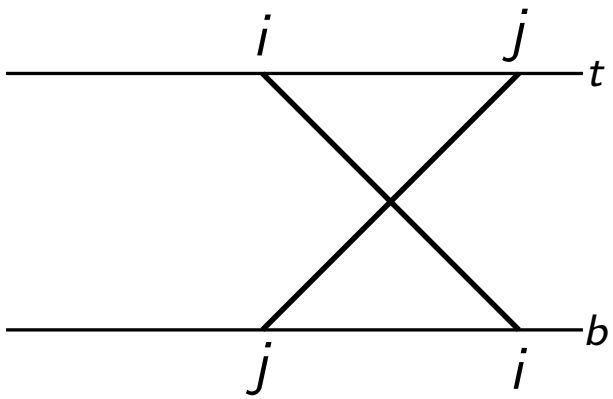
Crossing Pairs



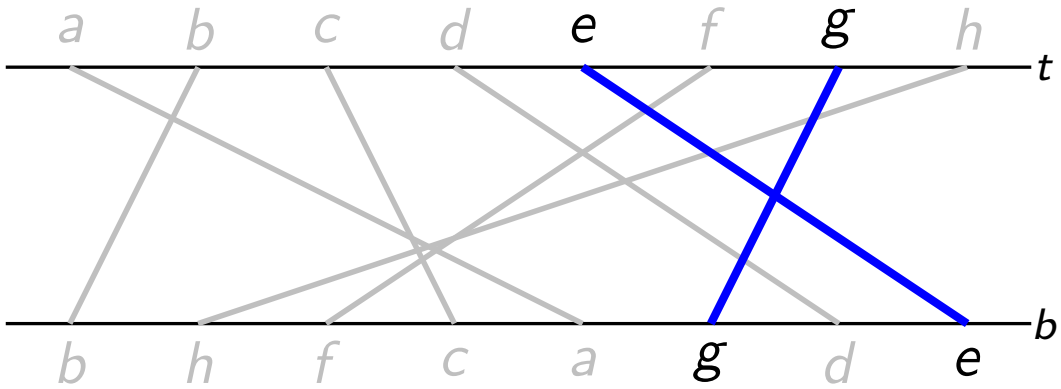
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

$$gh \leq_\ell ij \quad \Leftrightarrow \quad g \leq_t i \text{ and } h \leq_b j$$



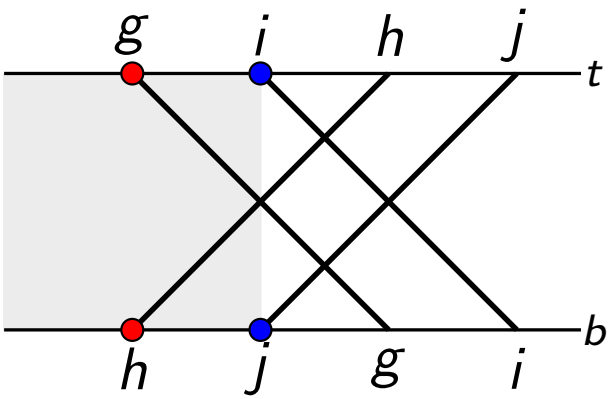


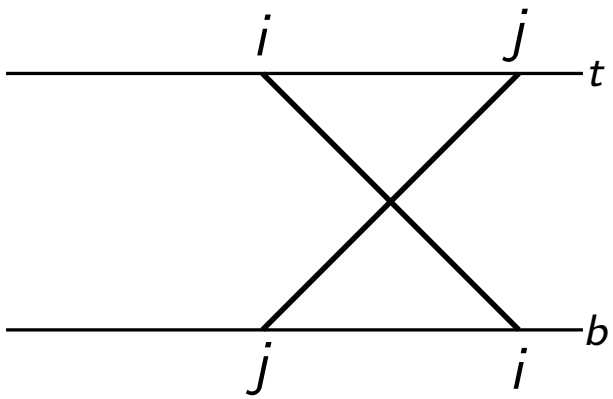
Crossing Pairs



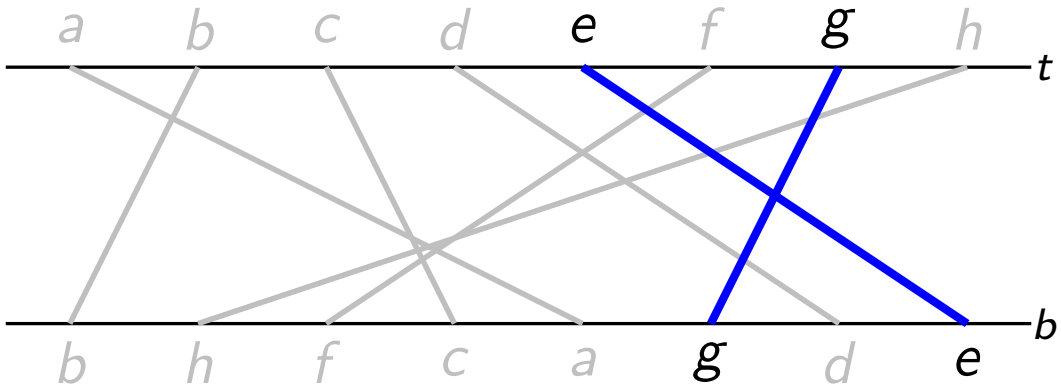
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

$$gh \leq_\ell ij \quad \Leftrightarrow \quad g \leq_t i \text{ and } h \leq_b j$$



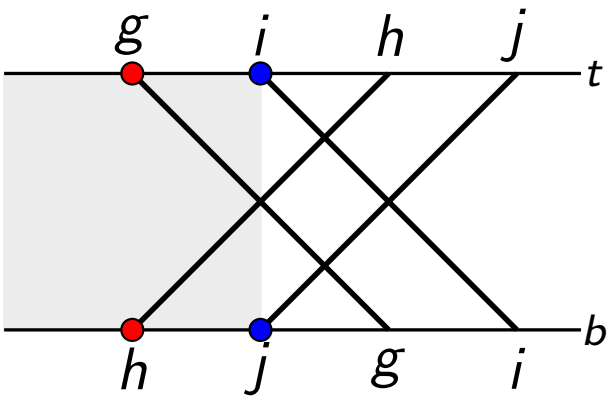


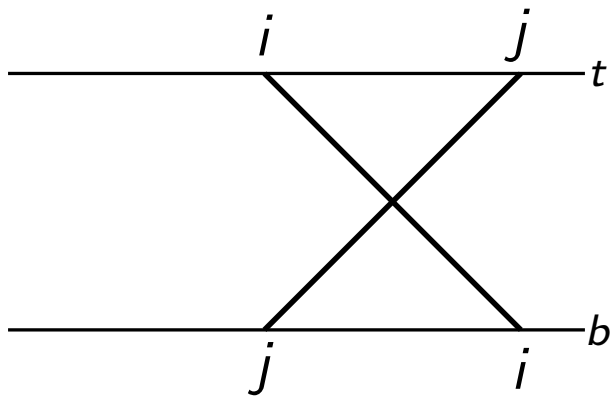
Crossing Pairs



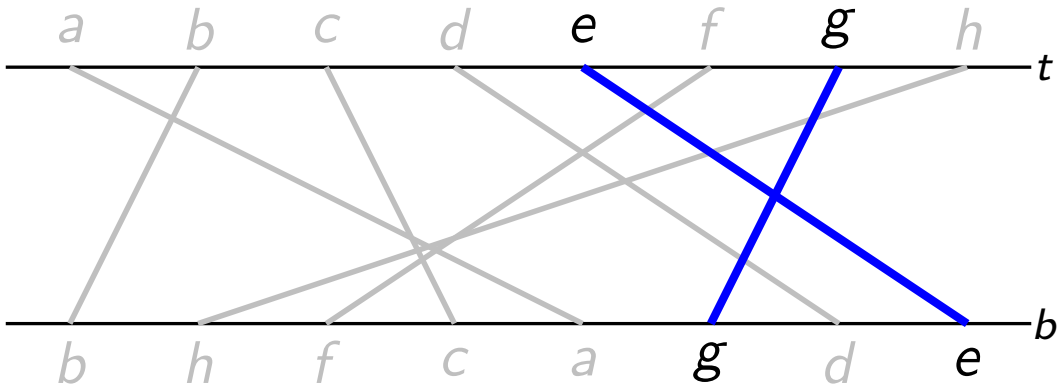
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

$$gh \leq_\ell ij \quad \Leftrightarrow \quad g \leq_t i \text{ and } h \leq_b j$$
$$gh \leq_r ij \quad \Leftrightarrow \quad g \leq_b i \text{ and } h \leq_t j$$



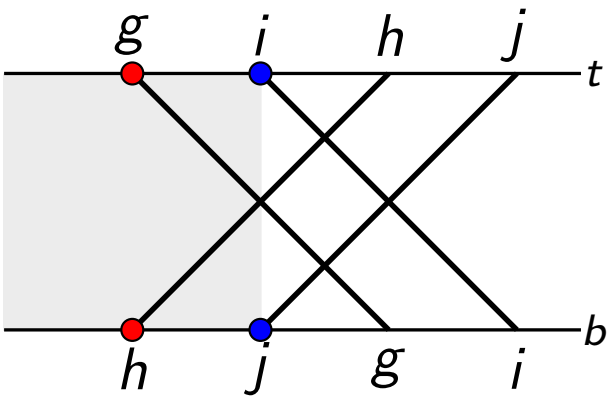


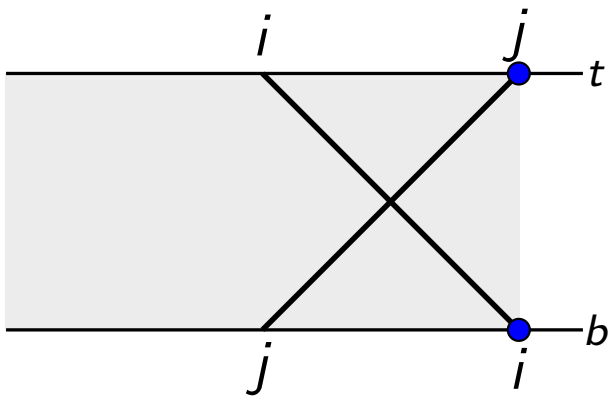
Crossing Pairs



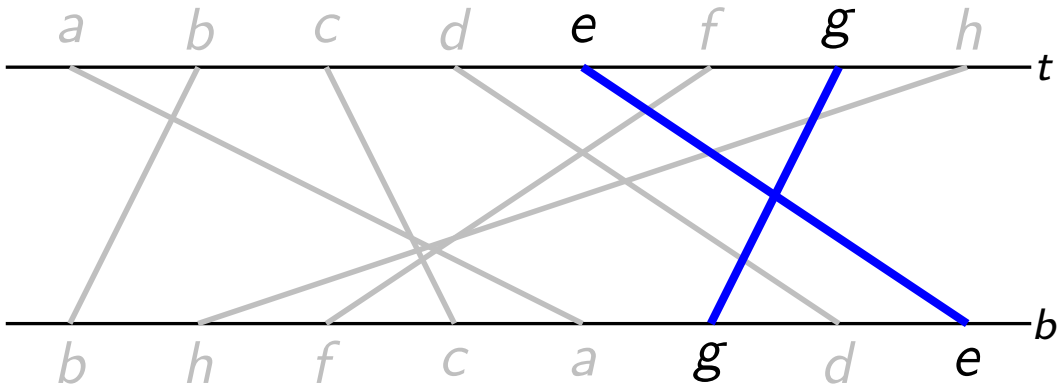
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

$$gh \leq_\ell ij \quad \Leftrightarrow \quad g \leq_t i \text{ and } h \leq_b j$$
$$gh \leq_r ij \quad \Leftrightarrow \quad g \leq_b i \text{ and } h \leq_t j$$



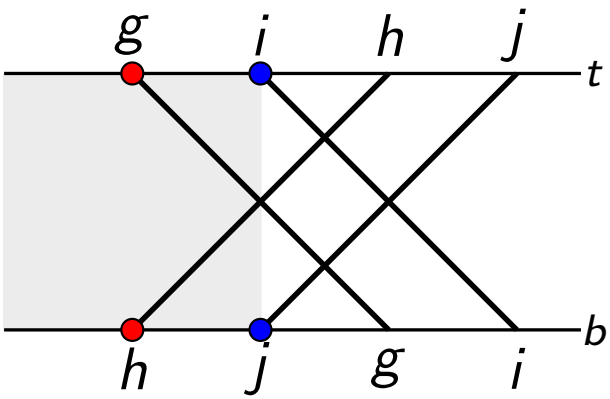


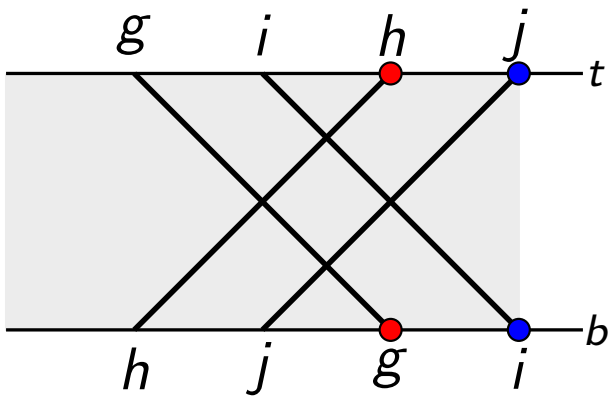
Crossing Pairs



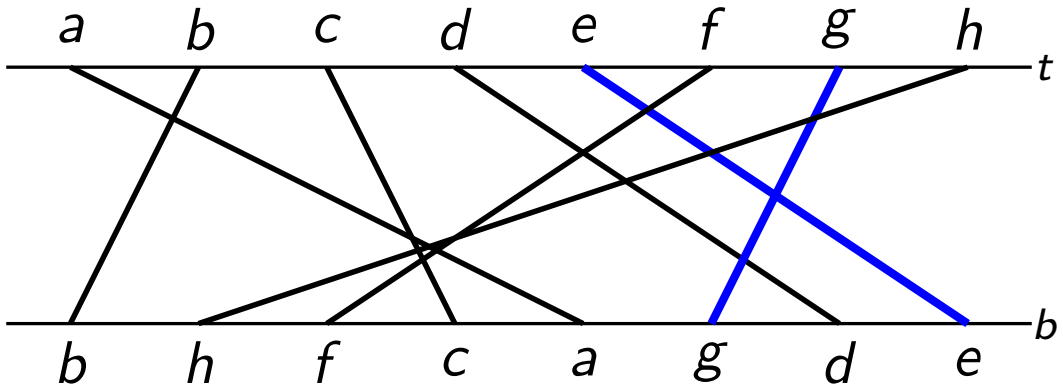
- $\mathcal{X}$: ij with $i \leq_t j$ and $j \leq_b i$
- Given $gh, ij \in \mathcal{X}$, we define $\leq_\ell$ and $\leq_r$:

$$gh \leq_\ell ij \quad \Leftrightarrow \quad g \leq_t i \text{ and } h \leq_b j$$
$$gh \leq_r ij \quad \Leftrightarrow \quad g \leq_b i \text{ and } h \leq_t j$$

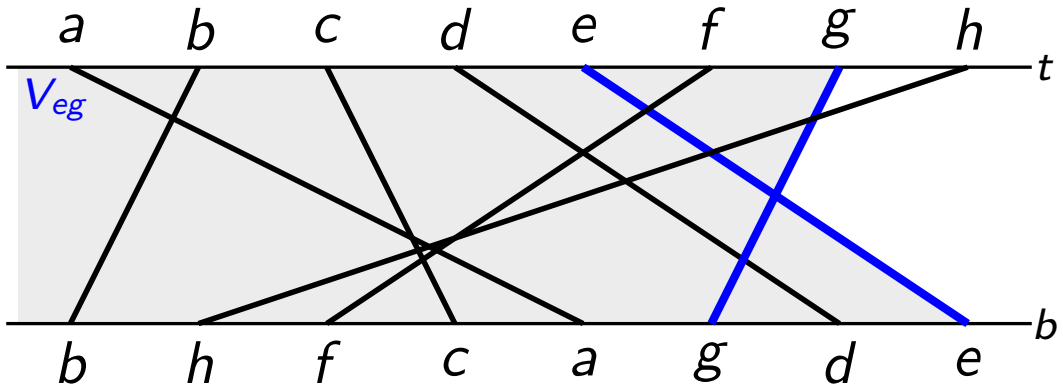




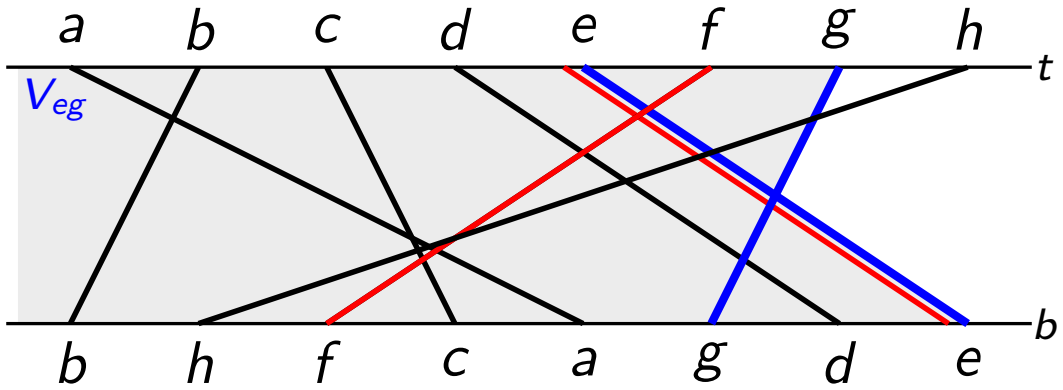
Predecessors of Crossing Pairs



Predecessors of Crossing Pairs

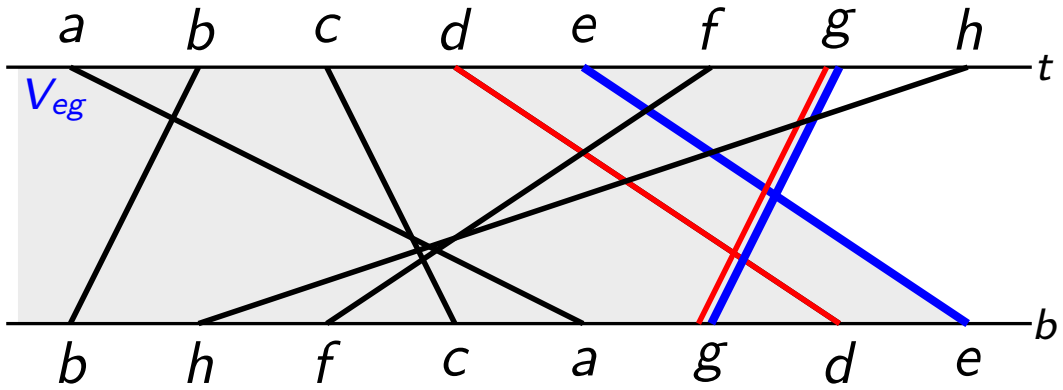


Predecessors of Crossing Pairs



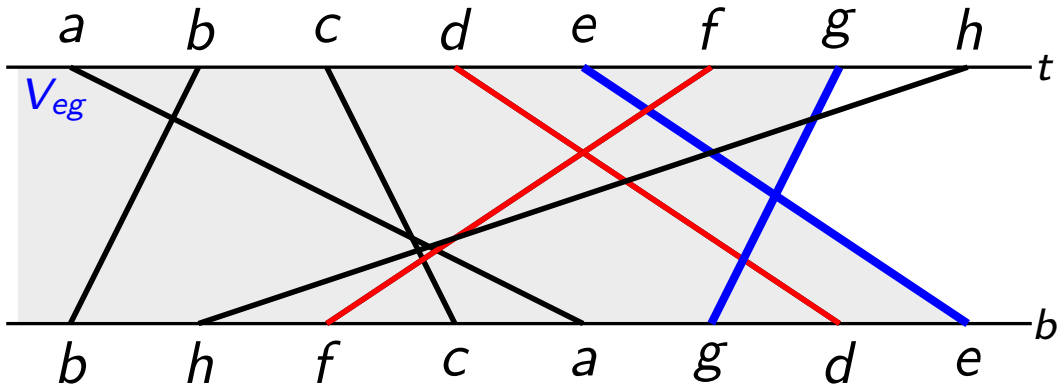
- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{j\}]$
  the iy with y the rightmost top

Predecessors of Crossing Pairs



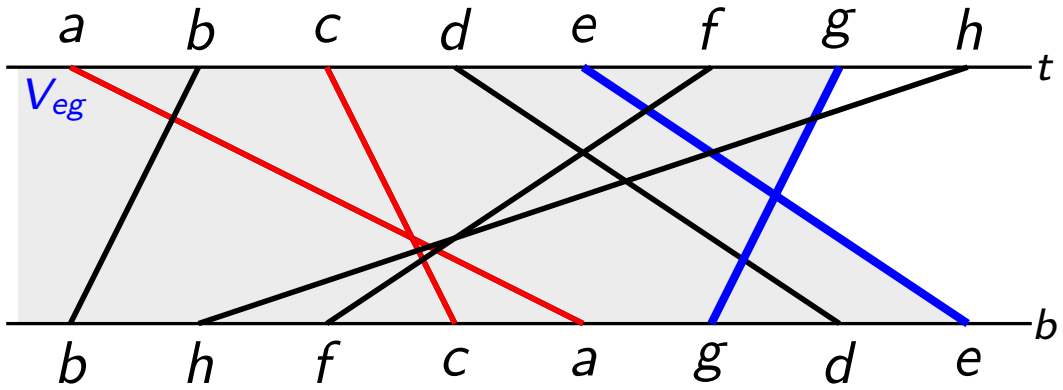
- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{j\}]$
  the iy with y the rightmost top
- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{i\}]$
  the xj with x the rightmost bottom

Predecessors of Crossing Pairs



- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{j\}]$
  the iy with y the rightmost top
- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{i\}]$
  the xj with x the rightmost bottom
- $< ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{i,j\}]$
  the xy with x and y the rightmost bottom and top

Predecessors of Crossing Pairs

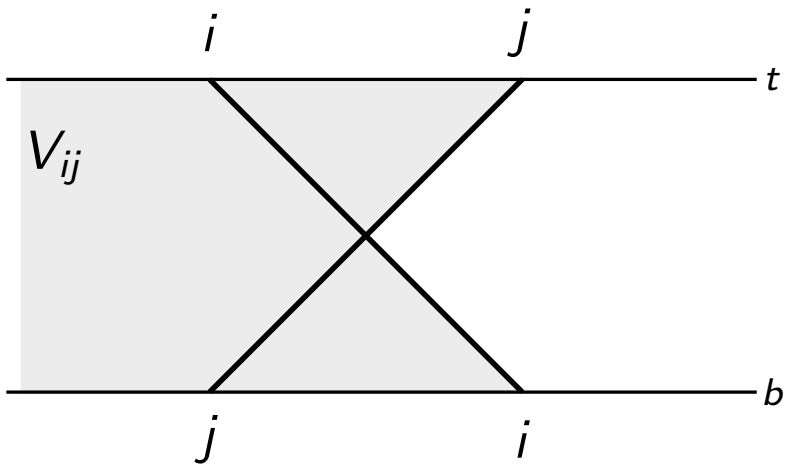


- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{j\}]$
  the iy with y the rightmost top
- $\leqslant ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{i\}]$
  the xj with x the rightmost bottom
- $< ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus \{i,j\}]$
  the xy with x and y the rightmost bottom and top
- $\ll ij = r\text{-}\max \mathcal{X}[V_{ij} \setminus (\{i,j\} \cup \{h \in V : \{h,i\} \in E \text{ or } \{h,j\} \in E\})]$
  non-adjacent xy with x and y the rightmost bottom and top

Basic sets: A, B, C

- Sets used by our dynamic programming algorithm
- A: corresponds to a maximum S-forest
- B and C: choose a solution only from a predescribed set
- $xy, zw \in V - V_{ij}$ such that $xy <_\ell zw$ and $\{x, w\}, \{y, z\} \in E$

$$A_{ij} = \max_w \{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

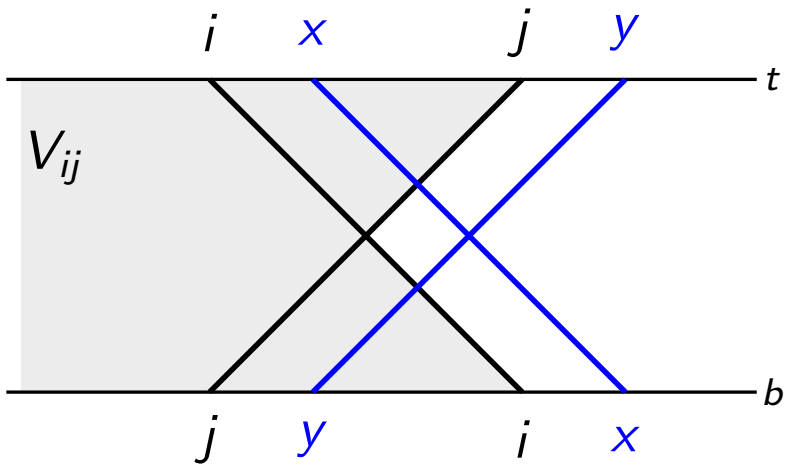


Basic sets: A, B, C

- Sets used by our dynamic programming algorithm
- A: corresponds to a maximum S-forest
- B and C: choose a solution only from a predescribed set
- $xy, zw \in V - V_{ij}$ such that $xy <_\ell zw$ and $\{x, w\}, \{y, z\} \in E$

$$A_{ij} = \max_w \{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w \{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$



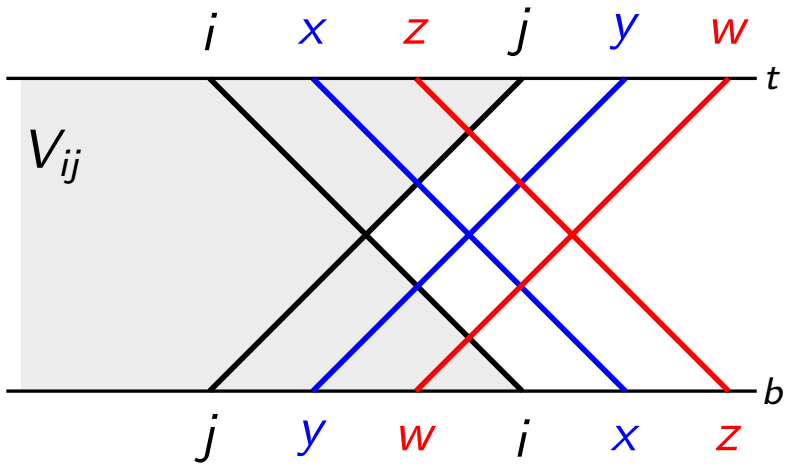
Basic sets: A, B, C

- Sets used by our dynamic programming algorithm
- A: corresponds to a maximum S-forest
- B and C: choose a solution only from a predescribed set
- $xy, zw \in V - V_{ij}$ such that $xy <_\ell zw$ and $\{x, w\}, \{y, z\} \in E$

$$A_{ij} = \max_w \{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w \{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w \{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$

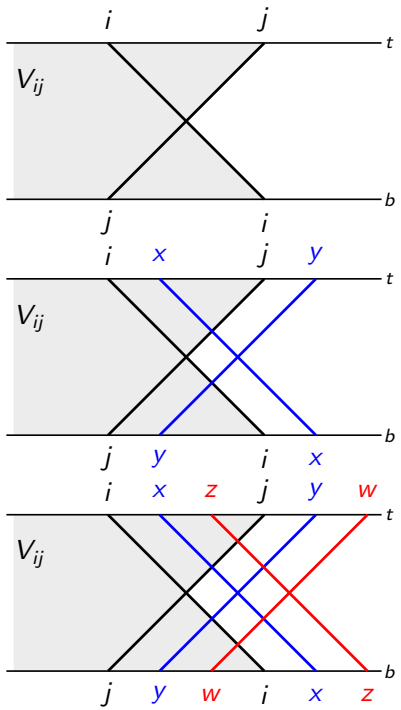


Recursive formulations for A, B, C

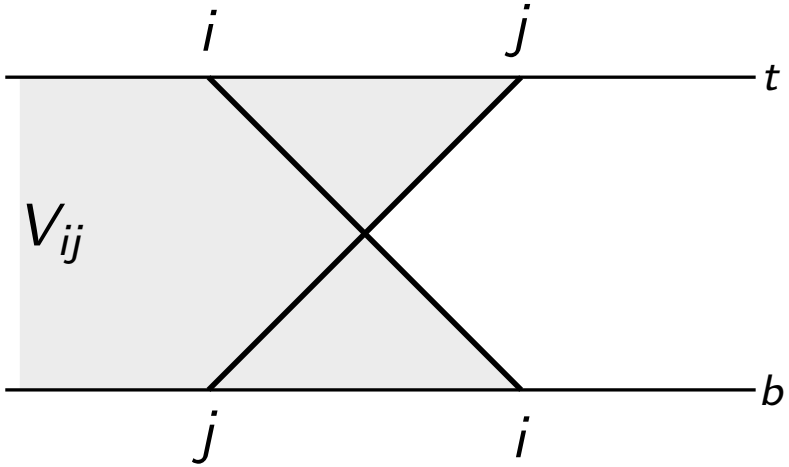
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$

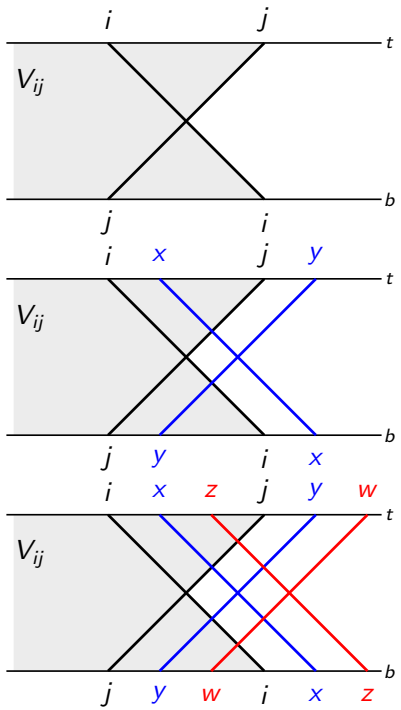


Recursive formulations for A, B, C

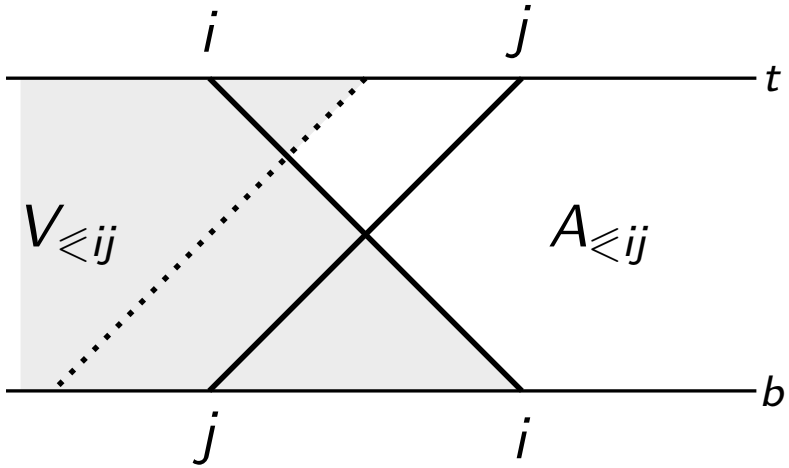
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$

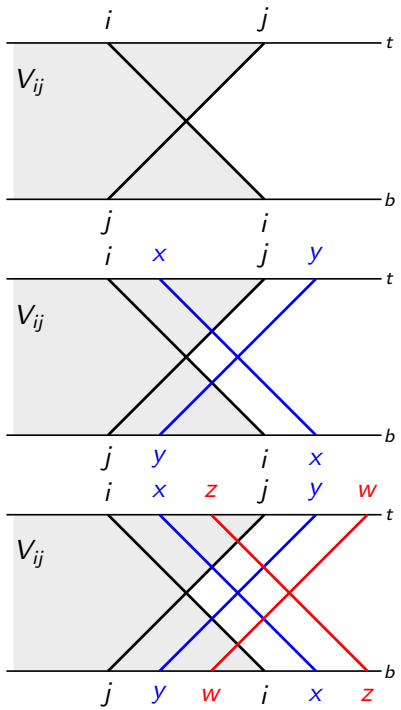


Recursive formulations for A, B, C

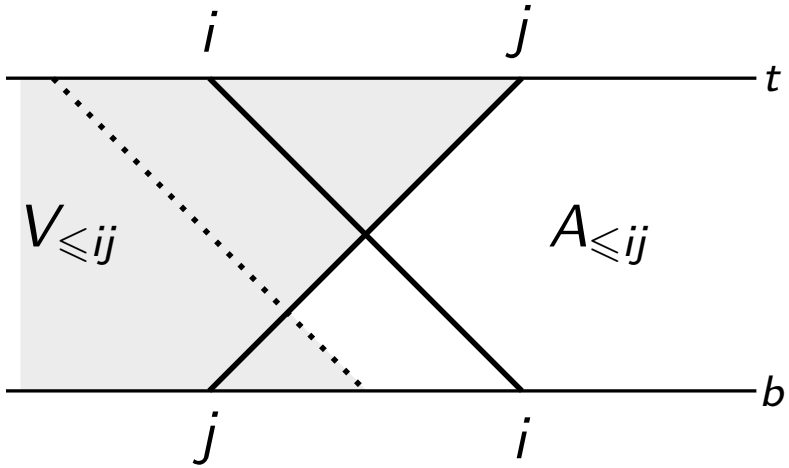
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$

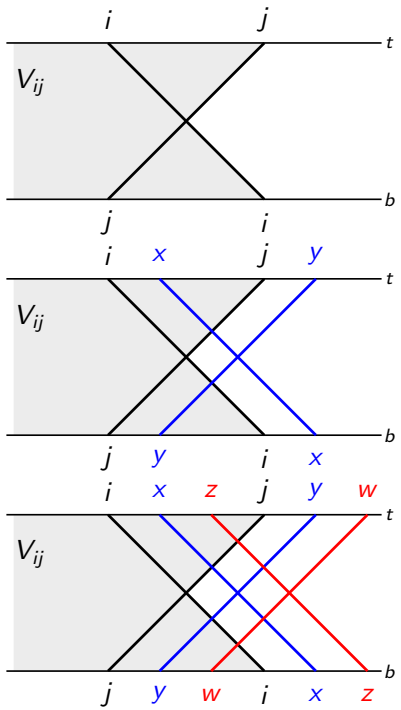


Recursive formulations for A, B, C

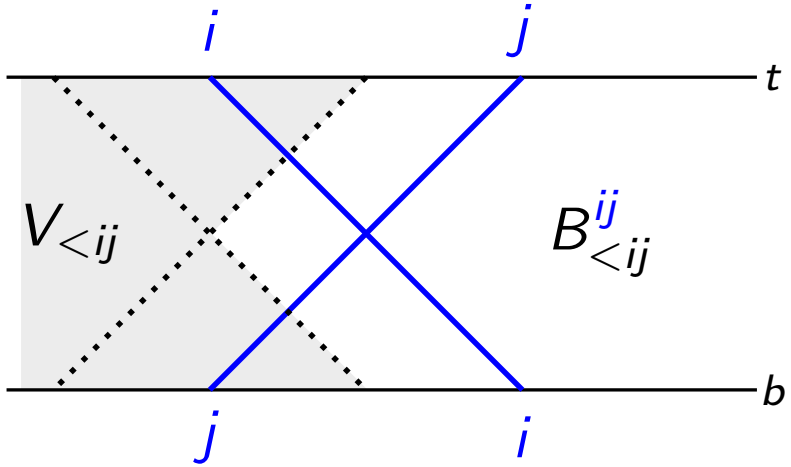
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$

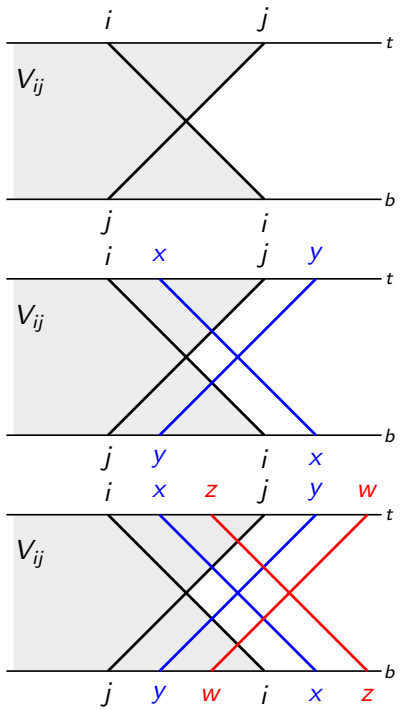


Recursive formulations for A, B, C

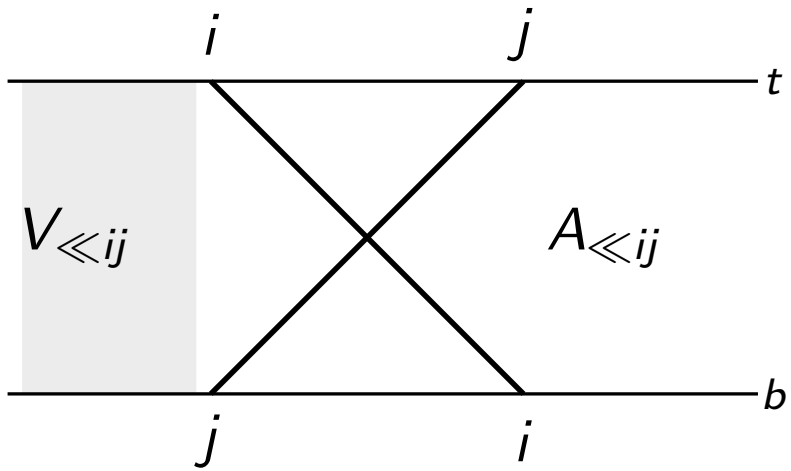
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$

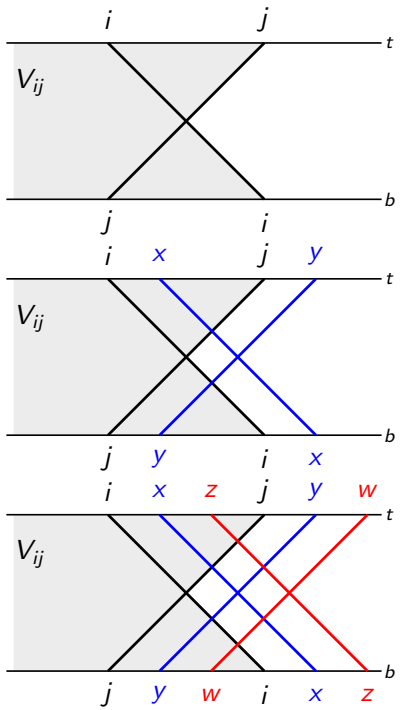


Recursive formulations for A, B, C

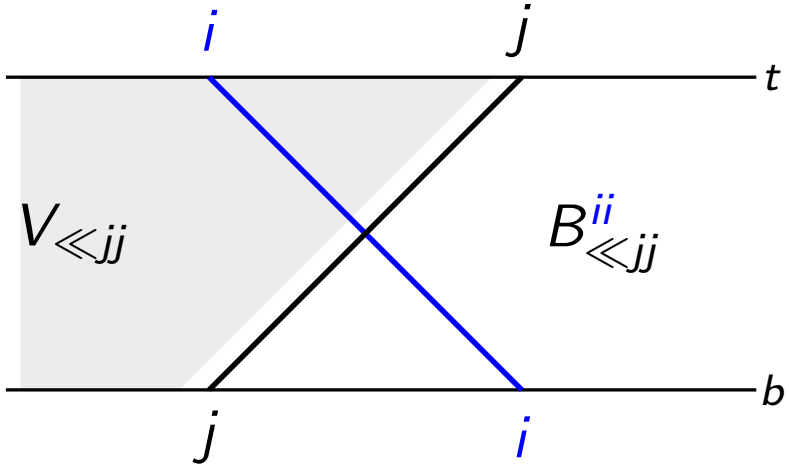
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$

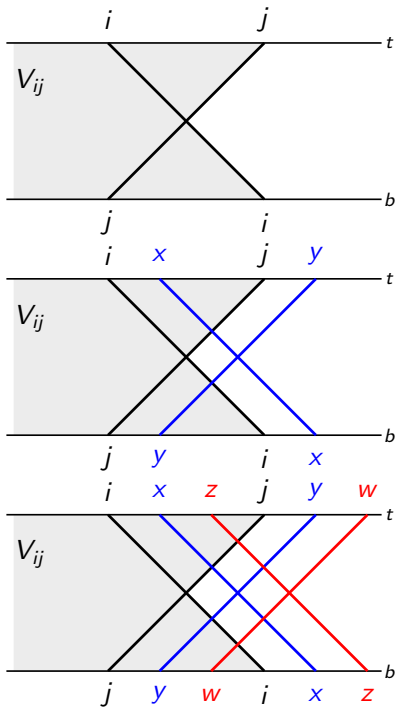


Recursive formulations for A, B, C

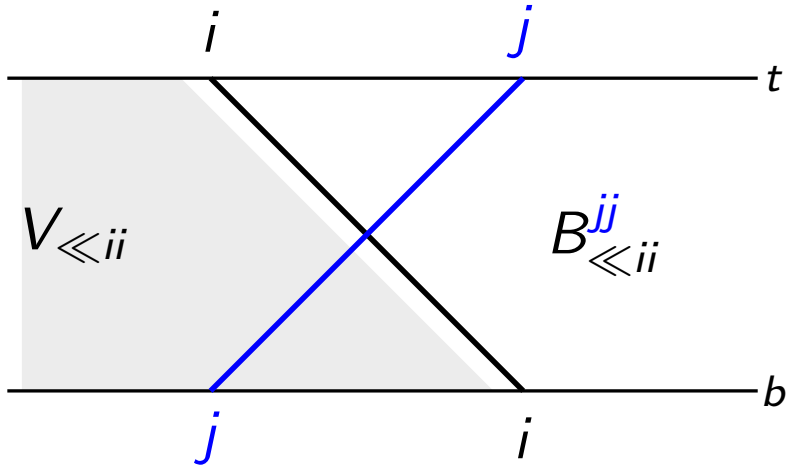
$$A_{ij} = \max_w\{X \subseteq V_{ij} : G[X] \in \mathcal{F}_S\}$$

$$B_{ij}^{xy} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y\}] \in \mathcal{F}_S\}$$

$$C_{ij}^{xy,zw} = \max_w\{X \subseteq V_{ij} : G[X \cup \{x, y, z, w\}] \in \mathcal{F}_S\}$$



The basic idea: we support any big S-cycle with a smaller S-cycle $\Rightarrow$



Poly-time for SFVS on Permutation Graphs

Theorem

There is a poly-time algorithm that computes SFVS of a permutation graph.

Poly-time for SFVS on Permutation Graphs

Theorem

There is a poly-time algorithm that computes SFVS of a permutation graph.

1. $\mathcal{X}$: there are $n + m$ crossing pairs

Poly-time for SFVS on Permutation Graphs

Theorem

There is a poly-time algorithm that computes SFVS of a permutation graph.

1. $\mathcal{X}$: there are $n + m$ crossing pairs
2. For each $ij \in \mathcal{X}$, we compute its $\{\lessdot, \leqslant, <, \ll\} \Rightarrow O(n^2 m)$

Poly-time for SFVS on Permutation Graphs

Theorem

There is a poly-time algorithm that computes SFVS of a permutation graph.

1. $\mathcal{X}$: there are $n + m$ crossing pairs
2. For each $ij \in \mathcal{X}$, we compute its $\{\lll, \leqslant, <, \ll\} \Rightarrow O(n^2 m)$
3. Scan all crossing pairs in an ascending order with respect to $<_r$:
   - First compute A_{ij}
   - Then compute B_{ij}^{xy} for every $xy \in V - V_{ij}$
   - And for each xy we compute $C_{ij}^{xy,zw}$ for every $zw \in V - V_{xy}$

Poly-time for SFVS on Permutation Graphs

Theorem

There is a poly-time algorithm that computes SFVS of a permutation graph.

1. $\mathcal{X}$: there are $n + m$ crossing pairs
2. For each $ij \in \mathcal{X}$, we compute its $\{\lll, \leqslant, <, \ll\} \Rightarrow O(n^2 m)$
3. Scan all crossing pairs in an ascending order with respect to $<_r$:
   - First compute A_{ij}
   - Then compute B_{ij}^{xy} for every $xy \in V - V_{ij}$
   - And for each xy we compute $C_{ij}^{xy,zw}$ for every $zw \in V - V_{xy}$
4. At the end, output $A_{\pi(n)n}$

Poly-time for SFVS on Permutation Graphs

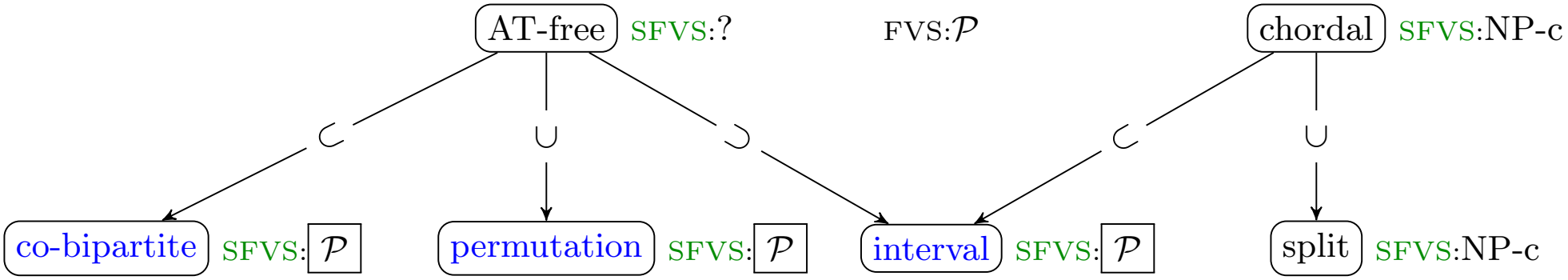
Theorem

There is a poly-time algorithm that computes SFVS of a permutation graph.

1. $\mathcal{X}$: there are $n + m$ crossing pairs
2. For each $ij \in \mathcal{X}$, we compute its $\{\lll, \leqslant, <, \ll\} \Rightarrow O(n^2 m)$
3. Scan all crossing pairs in an ascending order with respect to $<_r$:
   - First compute A_{ij}
   - Then compute B_{ij}^{xy} for every $xy \in V - V_{ij}$
   - And for each xy we compute $C_{ij}^{xy,zw}$ for every $zw \in V - V_{xy}$
4. At the end, output $A_{\pi(n)n}$

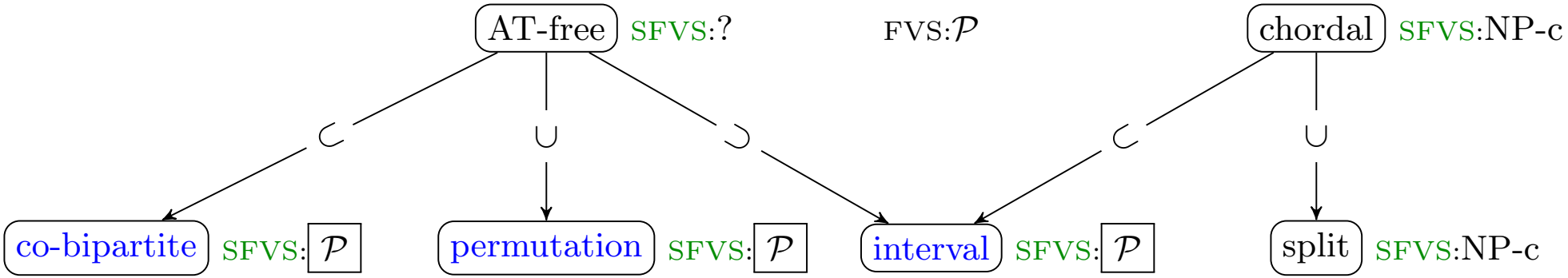
The total running time of the algorithm is $O(n + m^3)$.

Conclusion - Future work



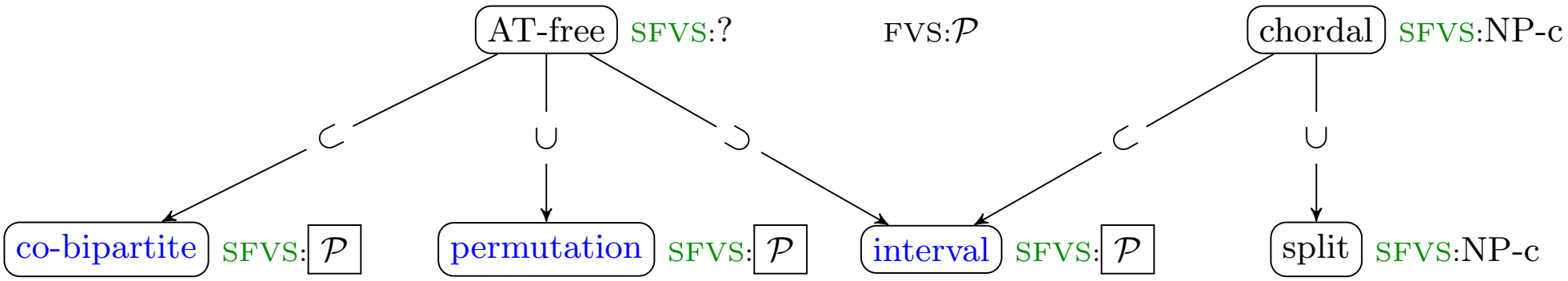
- Complexity on other graph classes:
  - ▷ AT-free graphs $\supset$ co-comparability graphs, I_3-free graphs
  - ▷ strongly chordal graphs, circular-arc graphs

Conclusion - Future work



- Complexity on other graph classes:
  - ▷ AT-free graphs $\supset$ co-comparability graphs, I_3-free graphs
  - ▷ strongly chordal graphs, circular-arc graphs
- Graphs of bounded structural parameter
  - ▷ bounded clique-width: FVS $\in \mathcal{P}$ (low exp-dep.) $\Rightarrow$ SFVS $\in$?
  - ▷ bounded induced matching width: due to the d.p. approach

Conclusion - Future work

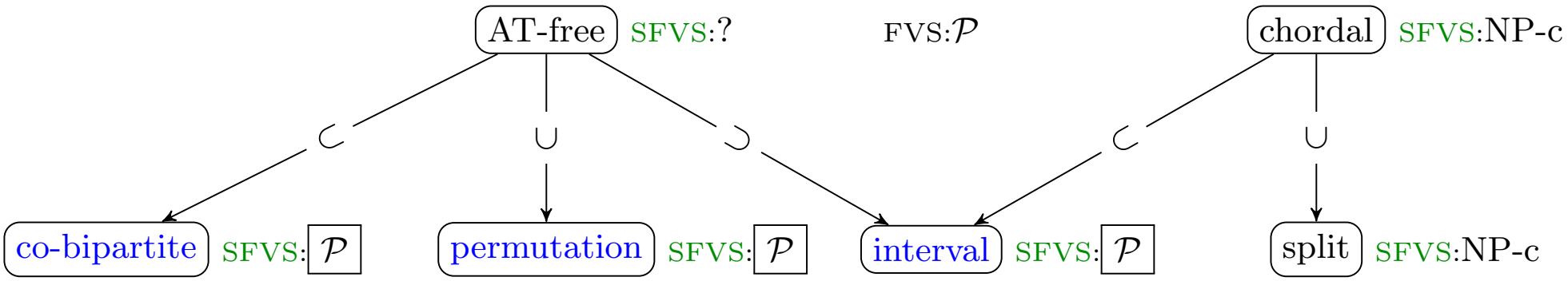


- Complexity on other graph classes:
  - ▷ AT-free graphs $\supset$ co-comparability graphs, I_3-free graphs
  - ▷ strongly chordal graphs, circular-arc graphs
- Graphs of bounded structural parameter
  - ▷ bounded clique-width: FVS $\in \mathcal{P}$ (low exp-dep.) $\Rightarrow$ SFVS $\in$?
  - ▷ bounded induced matching width: due to the d.p. approach

SFVS is related to terminal-sets problems:

- Multiway-Cut problem: disconnect a given set of terminals
- Can we adopt our algorithm on permutation or interval graphs in order to work for Multiway-Cut?

Conclusion - Future work



- Complexity on other graph classes:
  - ▷ AT-free graphs $\supset$ co-comparability graphs, I_3-free graphs
  - ▷ strongly chordal graphs, circular-arc graphs
- Graphs of bounded structural parameter
  - ▷ bounded clique-width: FVS $\in \mathcal{P}$ (low exp-dep.) $\Rightarrow$ SFVS $\in$?
  - ▷ bounded induced matching width: due to the d.p. approach

SFVS is related to terminal-sets problems:

- Multiway-Cut problem: disconnect a given set of terminals
- Can we adopt our algorithm on permutation or interval graphs in order to work for Multiway-Cut?

... Thank You!