On the structure of maximal cliques for Cocomparability Graphs
Maximal cliques para grafos de cocomparabilidad

Michel Habib

LAWCG, La plata 10 november 2016

Joint work with Barnaby Dalton, Derek G. Corneil, Jérémie Dusart and Ekkehard Khöler

From the articles:

- Derek Corneil, Barnaby Dalton and Michel Habib. LDFS-based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs. SIAM J. of Computing 42(3): 792-807 (2013).
- Derek G. Corneil, Jérémie Dusart, Michel Habib, and Ekkehard Khöler. On the power of graph searching for cocomparability graphs. SIAM Journal on Discrete Mathematics, 30(1):569 - 591, 2016.
- Jérémie Dusart, Michel Habib. A new LBFS-based algorithm for cocomparability graph recognition. Discrete Applied Mathematics to appear 2016. `http://dx.doi.org/10.1016/j.dam.2015.07.016`
- Jérémie Dusart, Michel Habib, Derek G. Corneil, On the structure of maximal cliques for Cocomparability Graphs and Applications, `http://arxiv.org/abs/1611.02002`

Work personally motivated by two questions

1. Why algorithms for interval graphs can be "generalized" or lifted to cocomparability graphs?

Work personally motivated by two questions

1. Why algorithms for interval graphs can be "generalized" or lifted to cocomparability graphs?
2. Why some of these algorithms based on graph searches look so greedy? What is the combinatorial structure involved?

Answer to questions

- Partial answers for the first question will be presented here, via a lattice structure of the maximal cliques.

Answer to questions

- Partial answers for the first question will be presented here, via a lattice structure of the maximal cliques.
- But for the second?

Comparability graphs

Comparability graph

A graph $G = (V, E)$ is a comparability graph if and only if G can be transitively oriented.

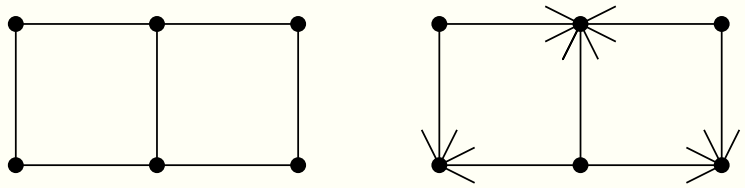


Figure: A comparability graph G and a transitive orientation of G.

Comparability graphs

Comparability graph

A graph $G = (V, E)$ is a comparability graph if and only if G can be transitively oriented.

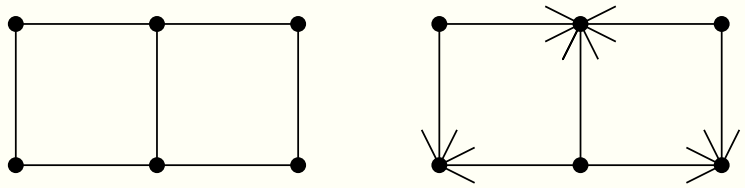


Figure: A comparability graph G and a transitive orientation of G.

Cocomparability graph

A graph $G = (V, E)$ is a cococomparability graph if and only if $\overline{G}$ is a comparability graph.

Not all graphs are comparability graphs

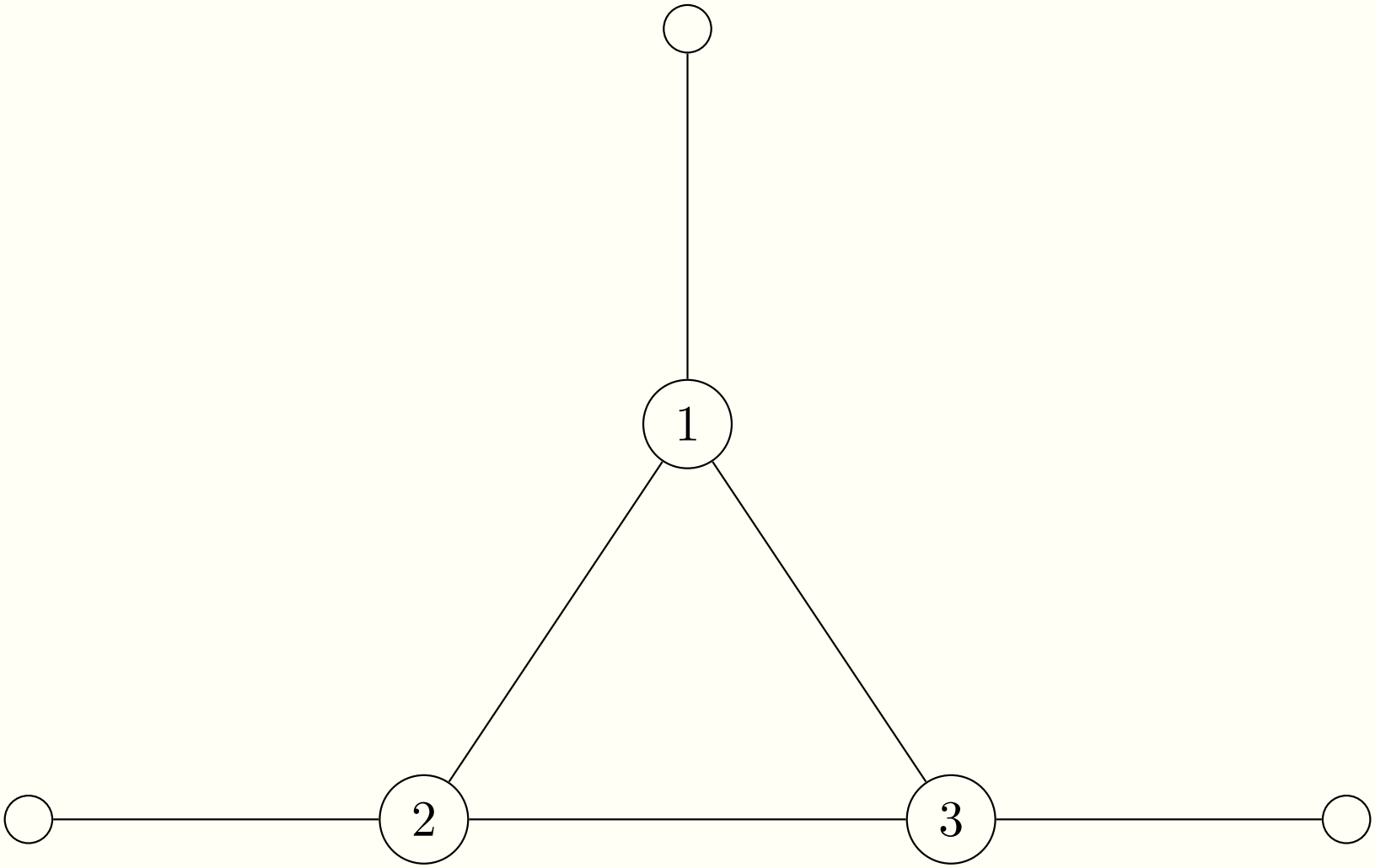


Figure: G is not a comparability graph

Not all graphs are comparability graphs

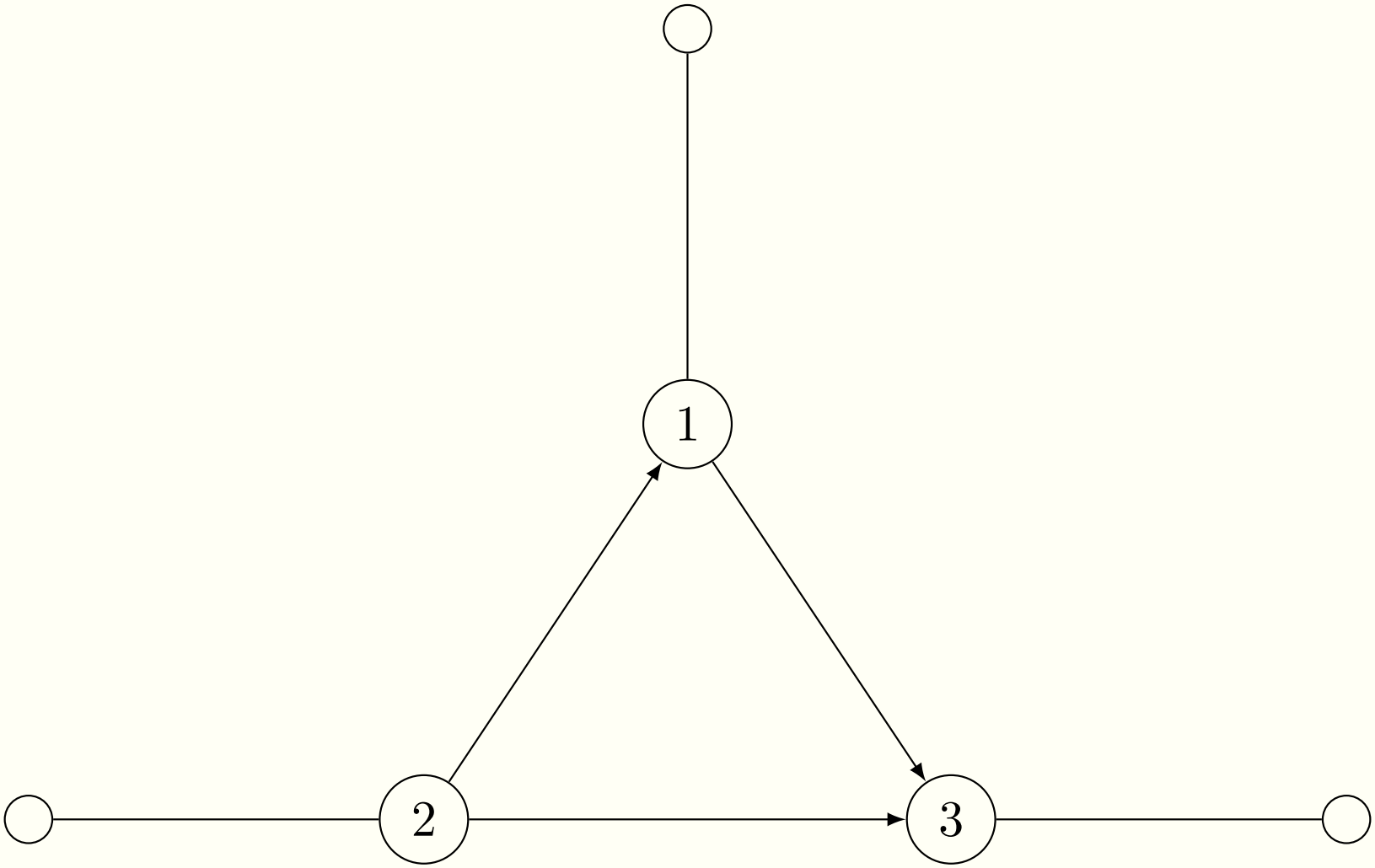
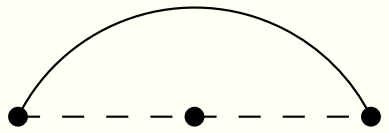


Figure: G is not a comparability graph

Cocomparability graphs

Definition:

For a total ordering τ of the set of vertices, an umbrella is a triple of vertices $a, b, c \in X$ such that: $a <_\tau b <_\tau c$ and $ac \in E$ and $ab, bc \notin E$.
A co-comparability (co-comp for short) ordering is an umbrella-free total ordering of the vertices of G.

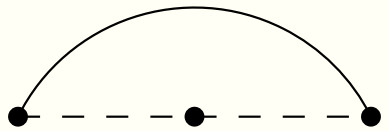


$a,\ b,\ c$, an umbrella

Cocomparability graphs

Definition:

For a total ordering τ of the set of vertices, an umbrella is a triple of vertices $a, b, c \in X$ such that: $a <_\tau b <_\tau c$ and $ac \in E$ and $ab, bc \notin E$.
A co-comparability (co-comp for short) ordering is an umbrella-free total ordering of the vertices of G.



$a,\ b,\ c$, an umbrella

Remark:

A cocomp ordering correspond to a linear extension of a transitive orientation of the complement.

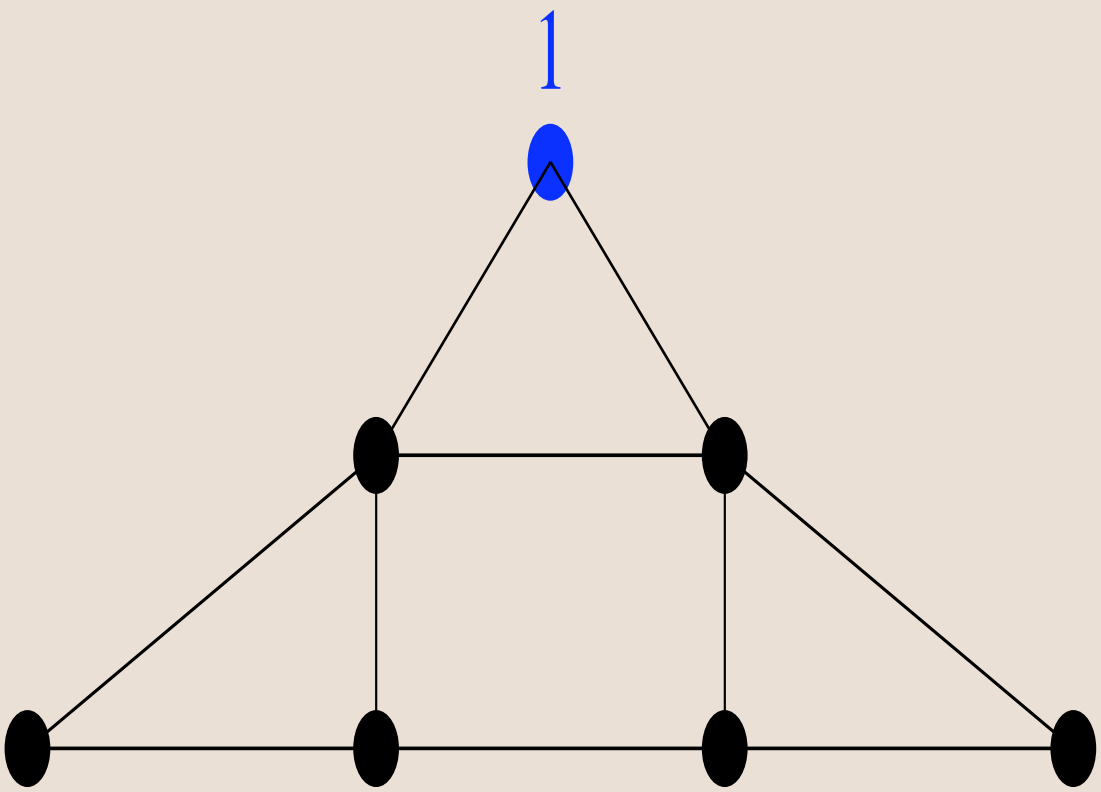
Graph searches

Graph Search

The graph is supposed to be connected so as the set of visited vertices. After choosing an initial vertex, a search of a connected graph visits each of the vertices and edges of the graph such that a new vertex is visited only if it is adjacent to some previously visited vertex.

At any point there may be several vertices that may possibly be visited next. To choose the next vertex we need a tie-break rule. The breadth-first search (BFS) and depth-first search (DFS) algorithms are the traditional strategies for determining the next vertex to visit.

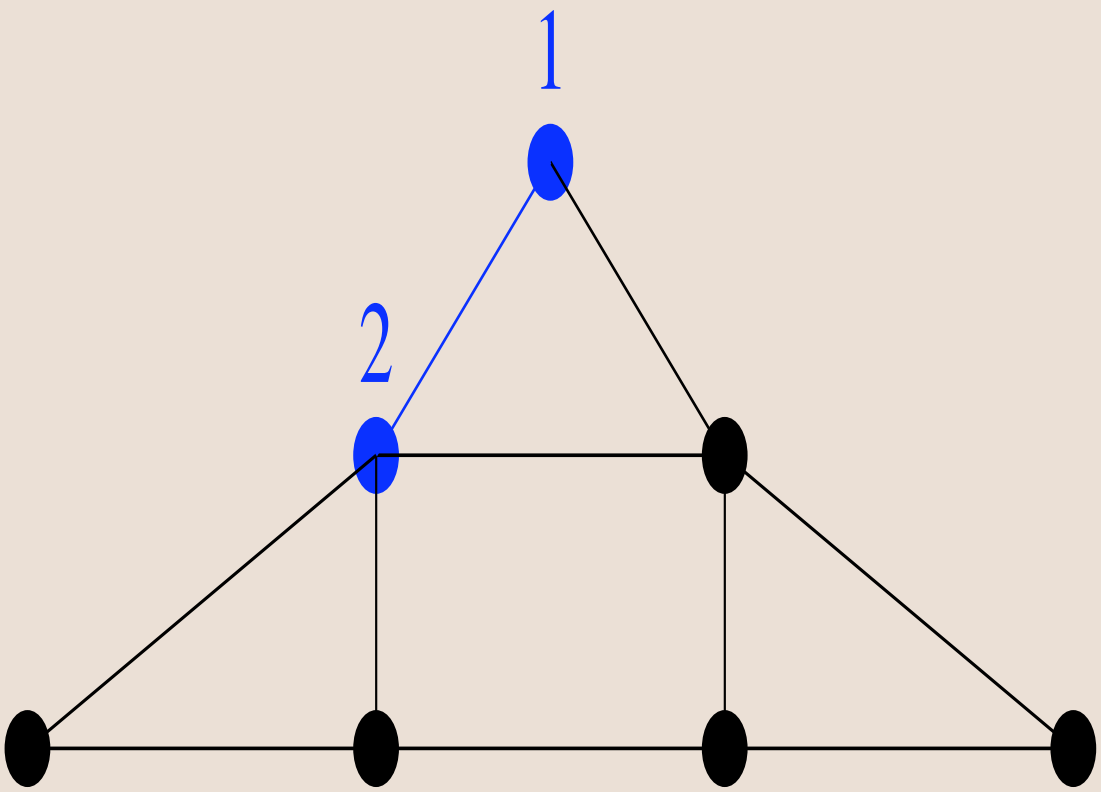
Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

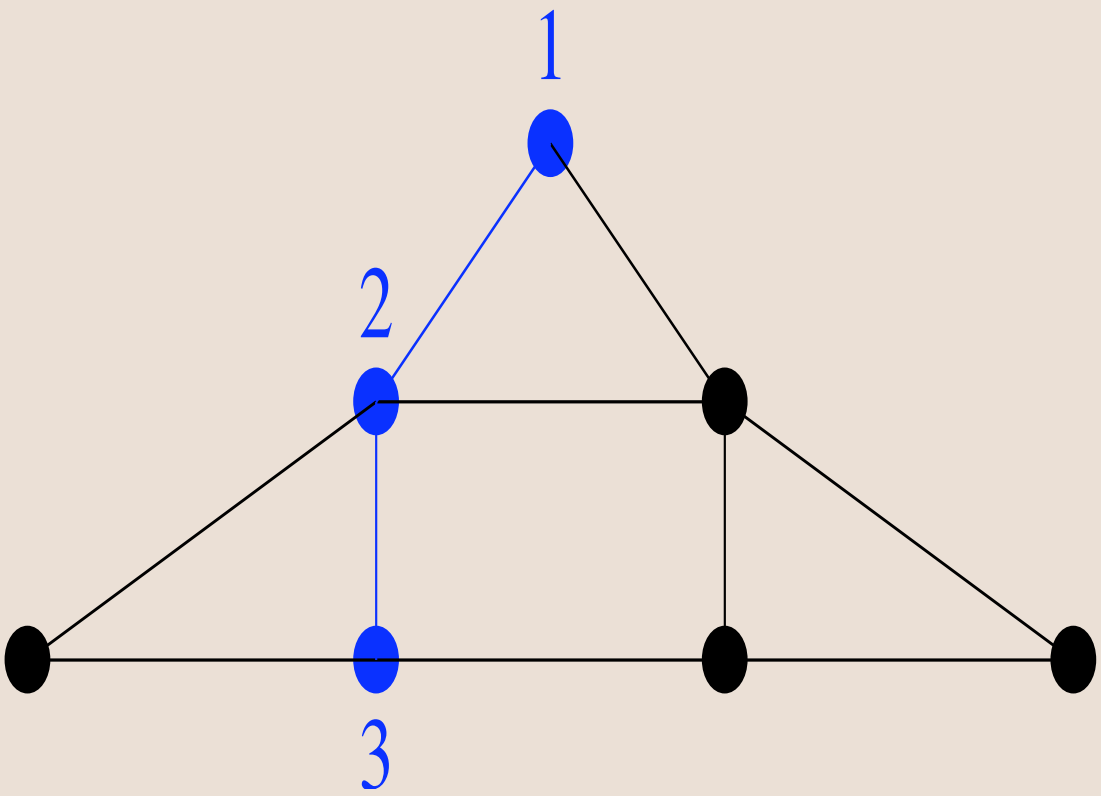
Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

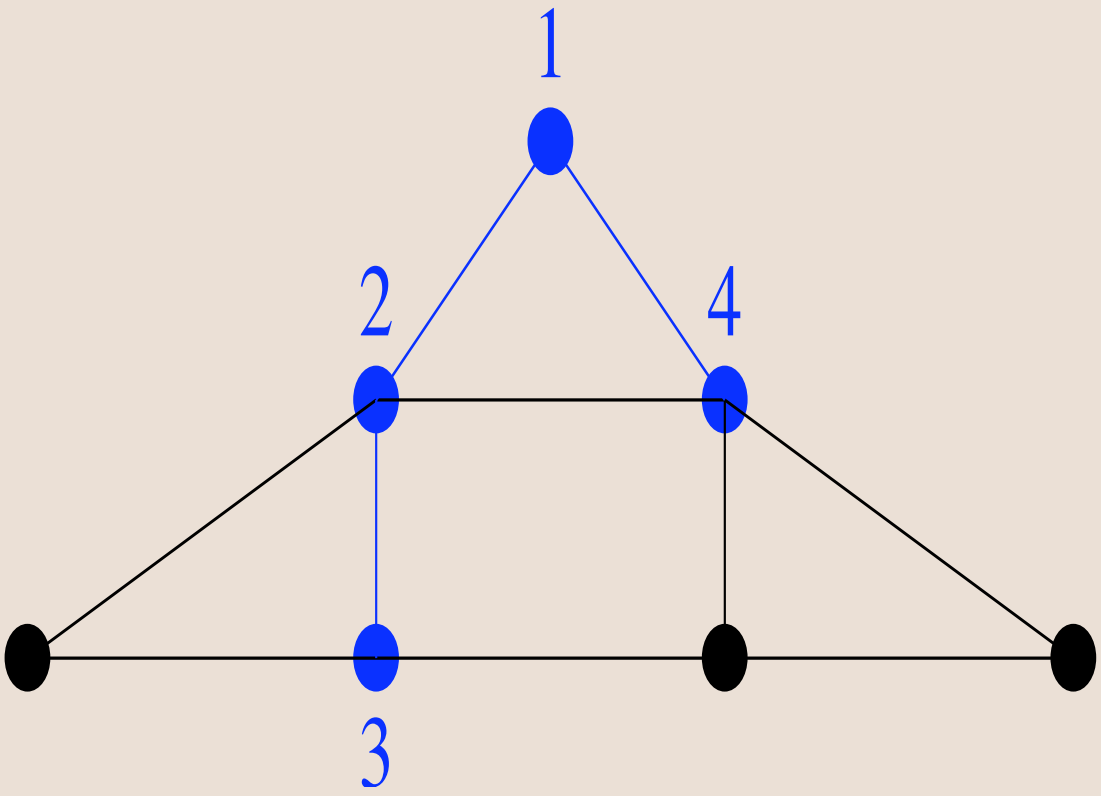
Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

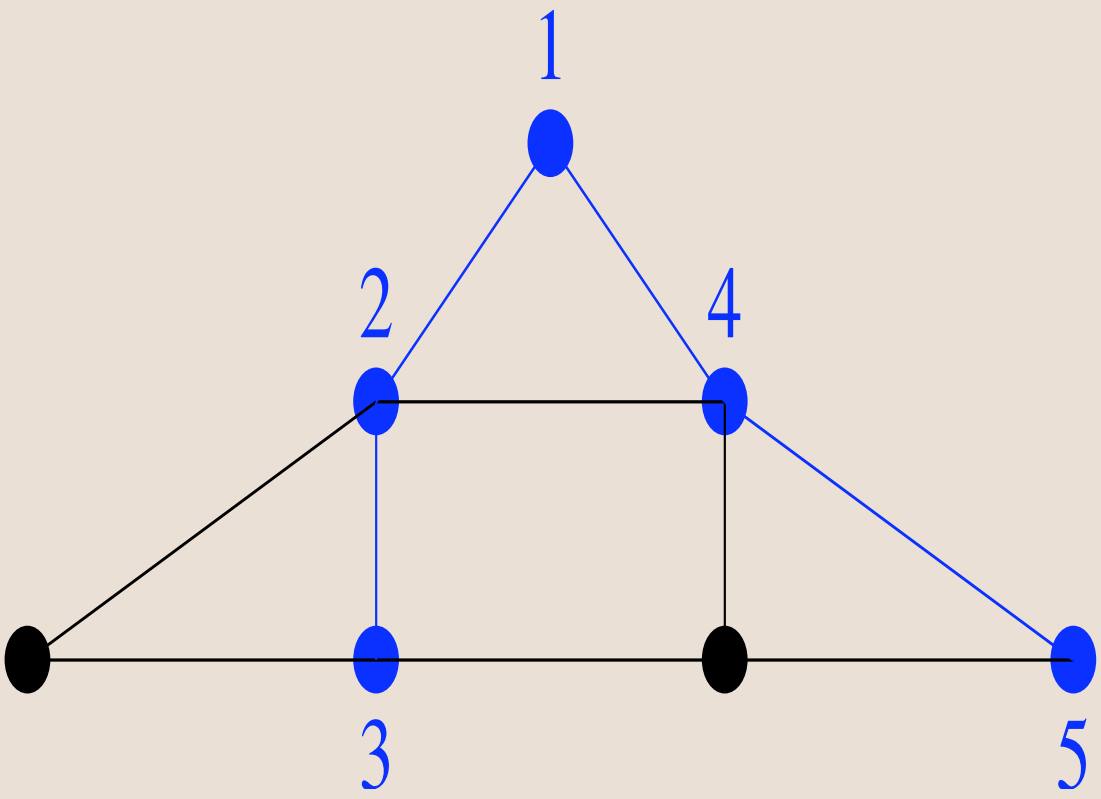
Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

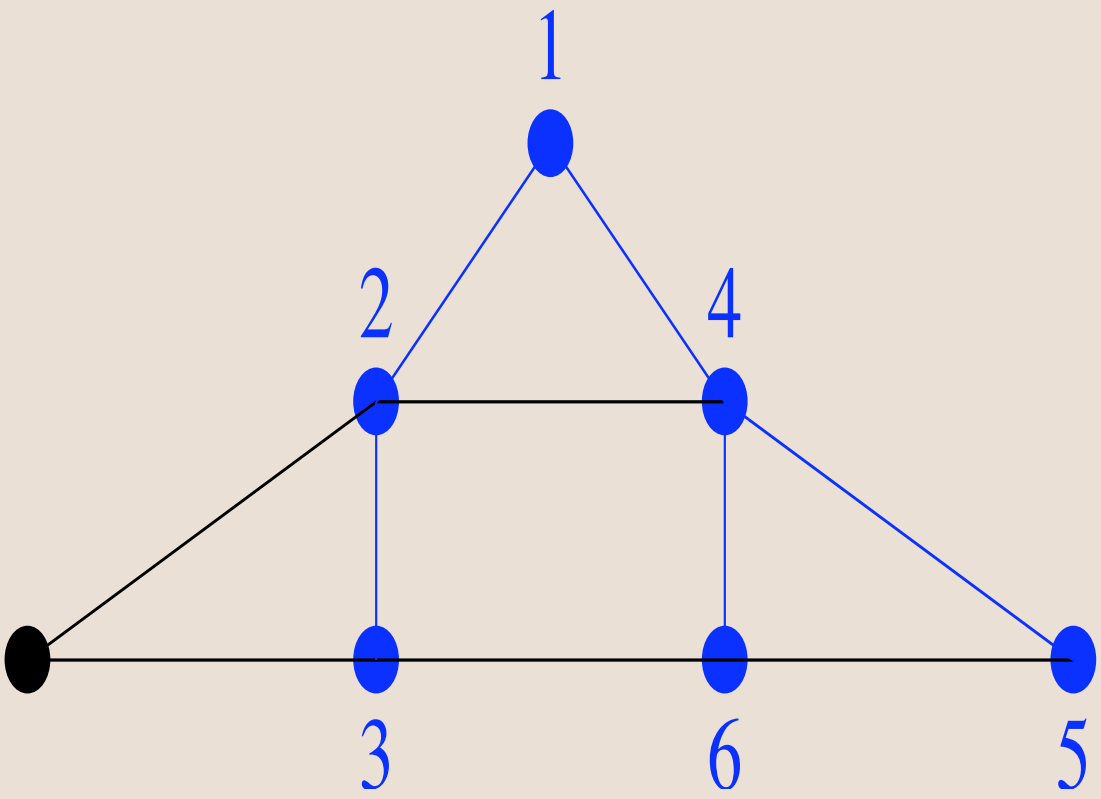
Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

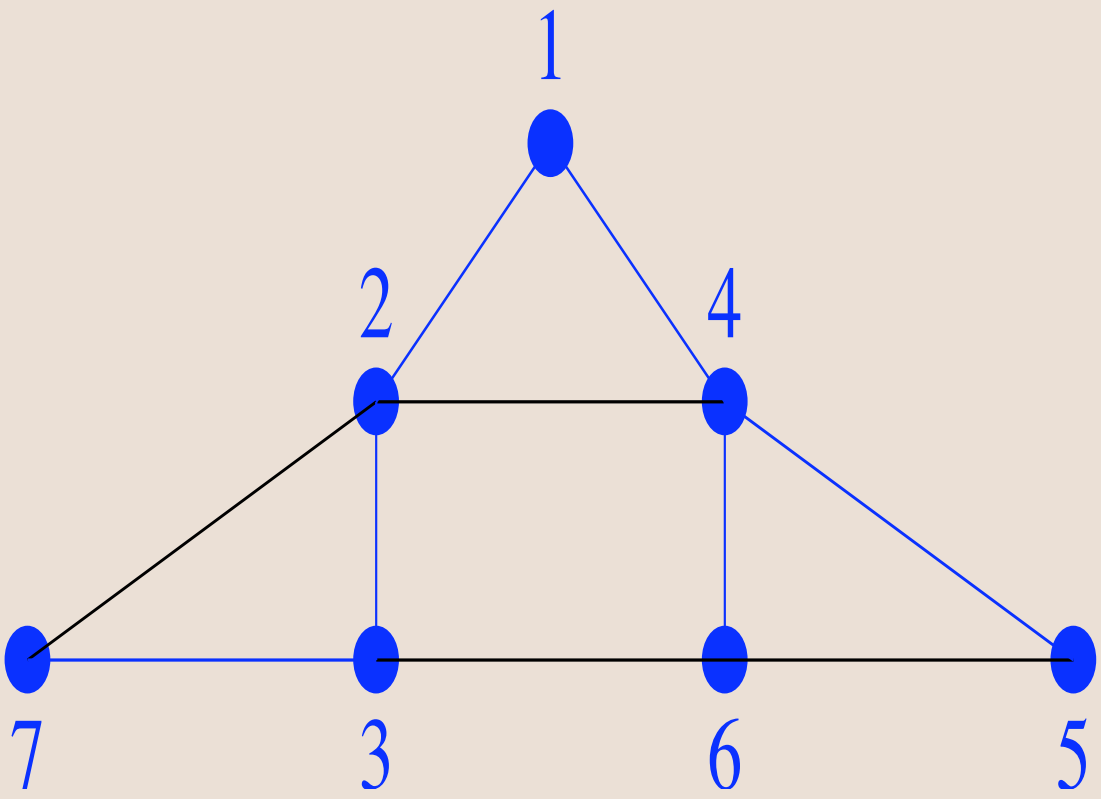
Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

Generic Search



Invariant

At each step, an edge between a visited vertex and a unvisited one is selected

Generic search

Generic search

$S \leftarrow \{s\}$;
for $i \leftarrow 1$ **to** n **do**
    Pick an unnumbered vertex v of S;
    $\sigma(i) \leftarrow v$;
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        if $w \notin S$ **then**
            Add w to S
        end
    end
end

The + tie-break rule

The + tie-break rule

At each step of the algorithm, the next vertex to be visited is the rightmost (or last) vertex of S in the ordering τ. S is the set of eligible vertices.
This + rule is due to K. Simon (1992). He used it in an "algorithm" to recognize interval graphs.

The + tie-break rule

The + tie-break rule

At each step of the algorithm, the next vertex to be visited is the rightmost (or last) vertex of S in the ordering τ. S is the set of eligible vertices.
This + rule is due to K. Simon (1992). He used it in an "algorithm" to recognize interval graphs.

Intuitively

This tool is used for multisweep graph searches. And the idea is to keep for tied vertices the ordering of the previous search.

The + tie-break rule

The + tie-break rule

At each step of the algorithm, the next vertex to be visited is the rightmost (or last) vertex of S in the ordering τ. S is the set of eligible vertices.
This + rule is due to K. Simon (1992). He used it in an "algorithm" to recognize interval graphs.

Intuitively

This tool is used for multisweep graph searches. And the idea is to keep for tied vertices the ordering of the previous search.

Characterization

We characterize graph searches for which cocomp orderings are closed under the + rule.

Corollary

For any cocomp ordering τ and any graph search $\mathcal{S}$ in $\{$Generic search, BFS, DFS, MNS, LBFS, LDFS, MCS, LocalMNS$\}$, $\sigma = \mathcal{S}^+(\mathcal{G}, \tau)$ is also a cocomp ordering.

Corollary

For any cocomp ordering τ and any graph search $\mathcal{S}$ in $\{$Generic search, BFS, DFS, MNS, LBFS, LDFS, MCS, LocalMNS$\}$, $\sigma = \mathcal{S}^+(\mathcal{G}, \tau)$ is also a cocomp ordering.

- This provides a nice toolbox to design special cocomp orderings.

Corollary

For any cocomp ordering τ and any graph search $\mathcal{S}$ in $\{$Generic search, BFS, DFS, MNS, LBFS, LDFS, MCS, LocalMNS$\}$, $\sigma = \mathcal{S}^+(\mathcal{G}, \tau)$ is also a cocomp ordering.

- This provides a nice toolbox to design special cocomp orderings.
- Could be useful for scheduling theory.

Minimum path cover for cocomp graphs with LDFS

- For a co-comparability graph G find a Minimum Path Cover

Minimum path cover for cocomp graphs with LDFS

- For a co-comparability graph G find a Minimum Path Cover
- Let P be a transitive orientation of $\overline{G}$
  our problem reduce to computing the bump number of P
  (Polynomial algorithm MH, Mohring, Steiner 1988)

Minimum path cover for cocomp graphs with LDFS

- For a co-comparability graph G find a Minimum Path Cover
- Let P be a transitive orientation of $\overline{G}$
  our problem reduce to computing the bump number of P
  (Polynomial algorithm MH, Mohring, Steiner 1988)
- Another equivalent formulation as the 2-machine scheduling problem
  (Another polynomial algorithm Gabow, Tarjan 1985)

Lexicographic Depth-First Search

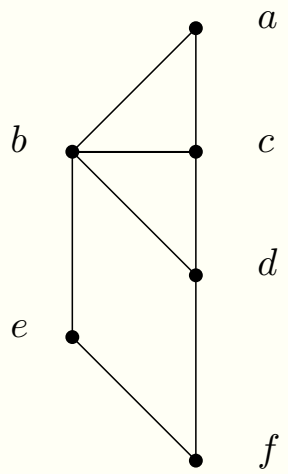
LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



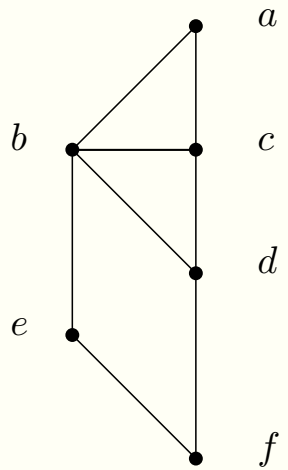
vertex	label
d	ϵ
c	ϵ
b	ϵ
f	ϵ
a	ϵ
e	ϵ

$i = 1,\ \sigma =$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



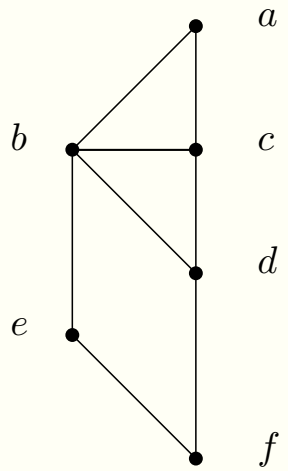
vertex	label
c	ϵ
b	ϵ
f	ϵ
a	ϵ
e	ϵ

$i = 1,\ \sigma = d$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



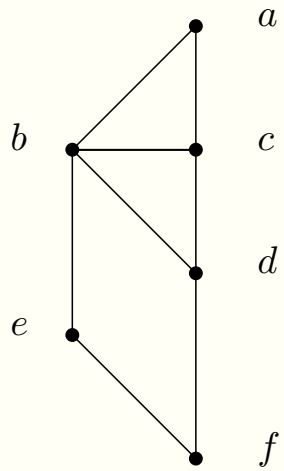
vertex	label
c	1
b	1
f	1
a	ϵ
e	ϵ

$i = 2,\ \sigma = d$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



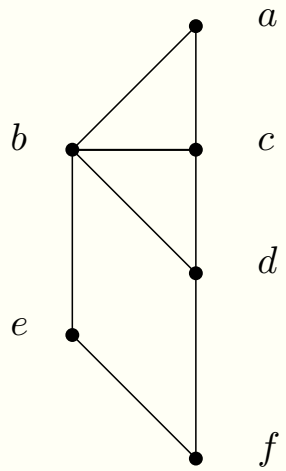
vertex	label
b	1
f	1
a	ϵ
e	ϵ

$i = 2,\ \sigma = d,\ c$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



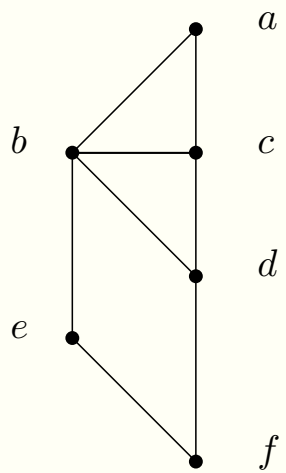
vertex	label
b	21
a	2
f	1
e	ϵ

$i = 3,\ \sigma = d,\ c$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



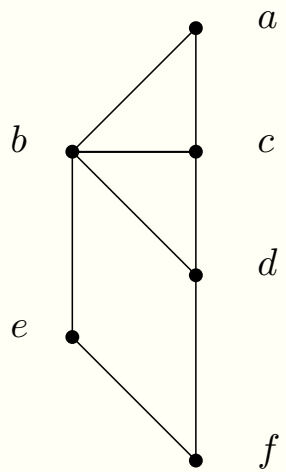
vertex	label
a	2
f	1
e	ϵ

$i = 3,\ \sigma = d,\ c,\ b$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



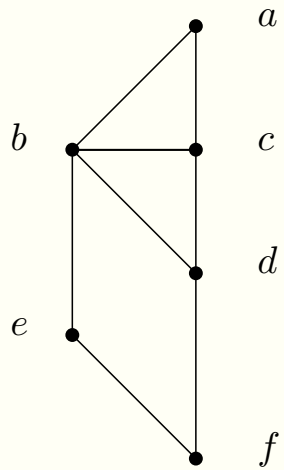
vertex	label
a	32
e	3
f	1

$i = 4,\ \sigma = d,\ c,\ b$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



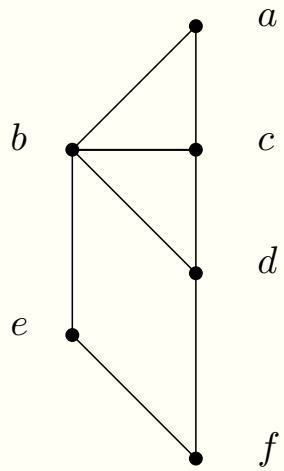
vertex	label
e	3
f	1

$i = 4,\ \sigma = d,\ c,\ b,\ a$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



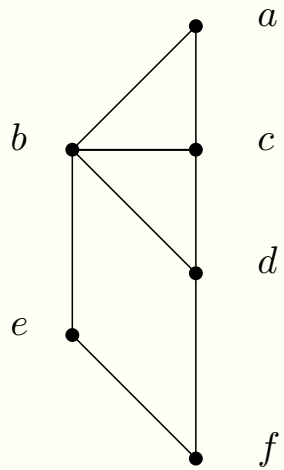
vertex	label
e	3
f	1

$i = 5,\ \sigma = d,\ c,\ b,\ a$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



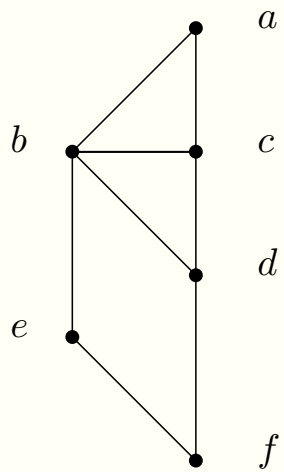
vertex	label
f	1

$i = 5,\ \sigma = d,\ c,\ b,\ a,\ e$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



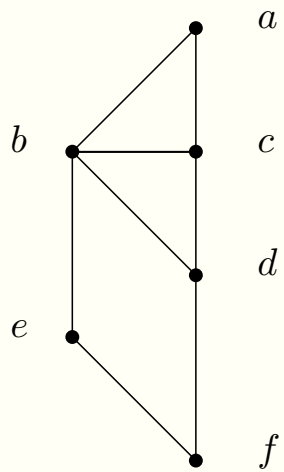
vertex	label
f	61

$i = 6,\ \sigma = d,\ c,\ b,\ a,\ e$

Lexicographic Depth-First Search

LDFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Prepend letter i to $label(u)$



vertex	label

$i = 6, \sigma = d,\ c,\ b,\ a,\ e,\ f$

First algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)

First algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.

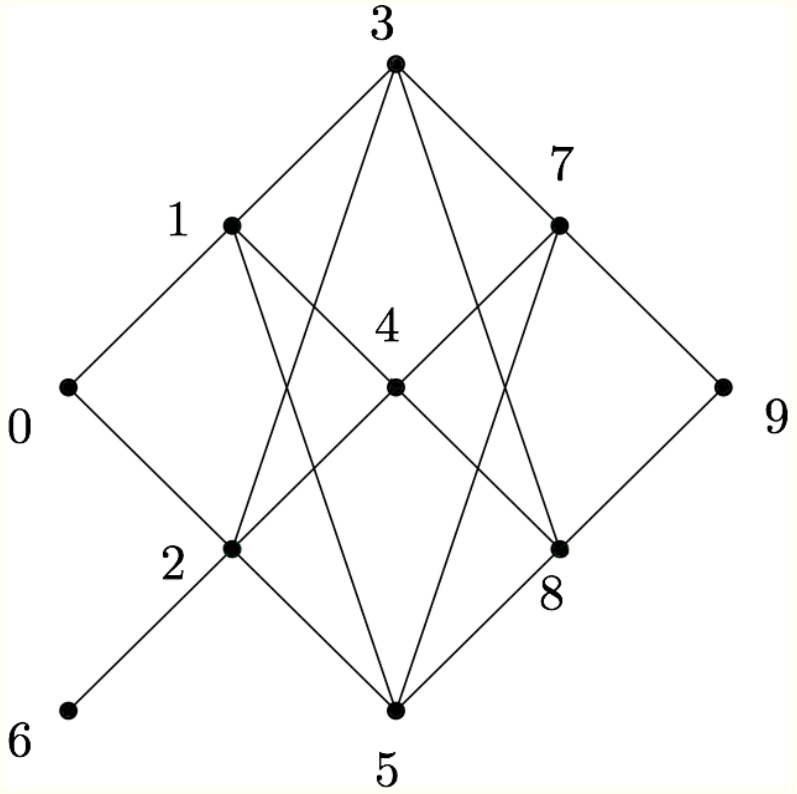
First algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.
3. Apply $RightMostNeighbour(\tau)$ which gives the path cover

First algorithm

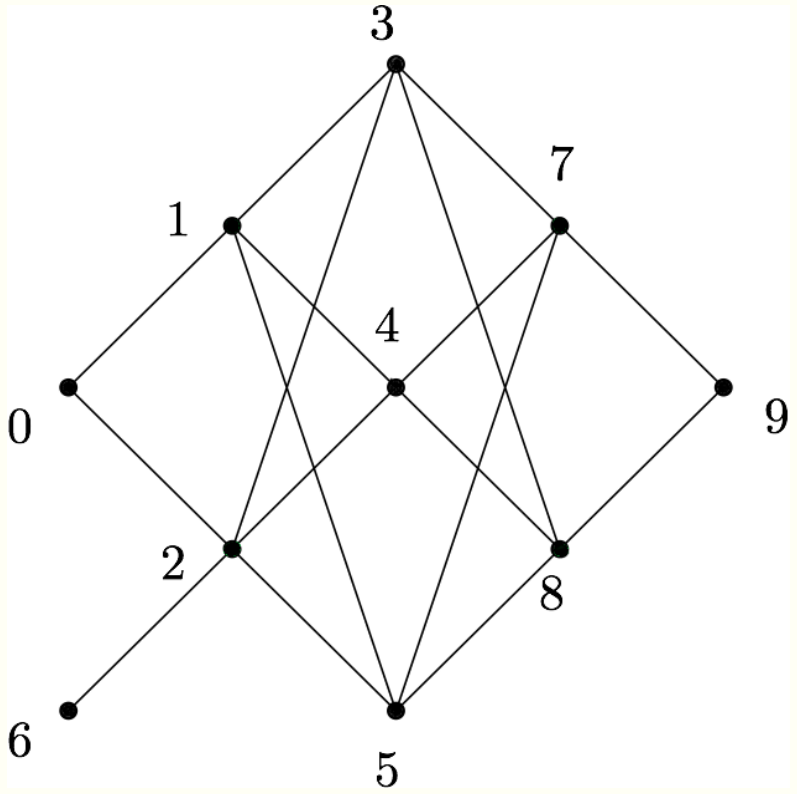
1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^+(\sigma)$ to produce an ordering τ.
3. Apply $RightMostNeighbour(\tau)$ which gives the path cover
4. Exhibit a certificate of minimality with a cut-set.

Let us take an example



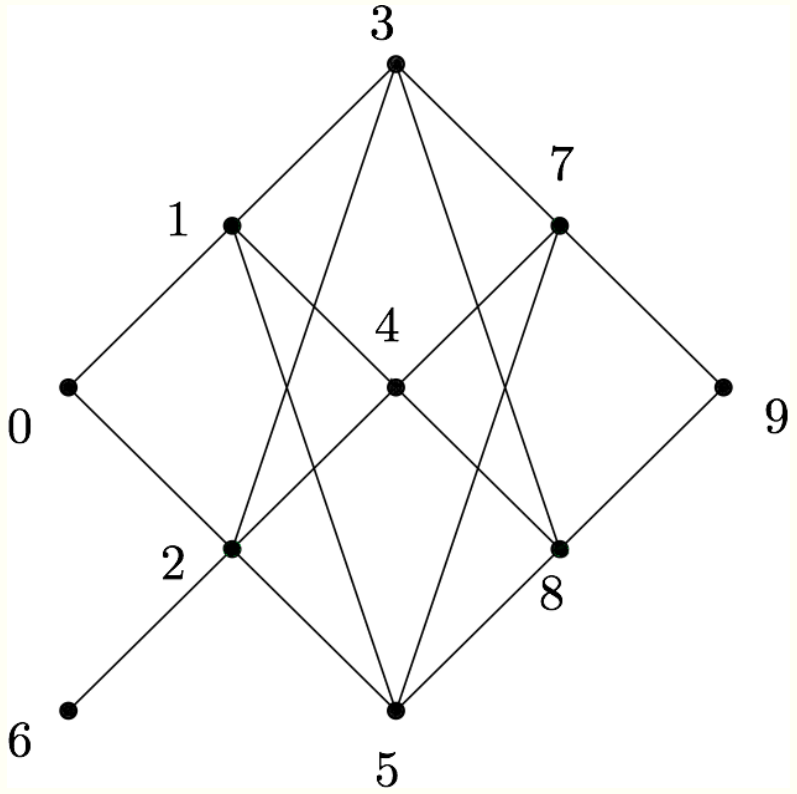
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering

Let us take an example



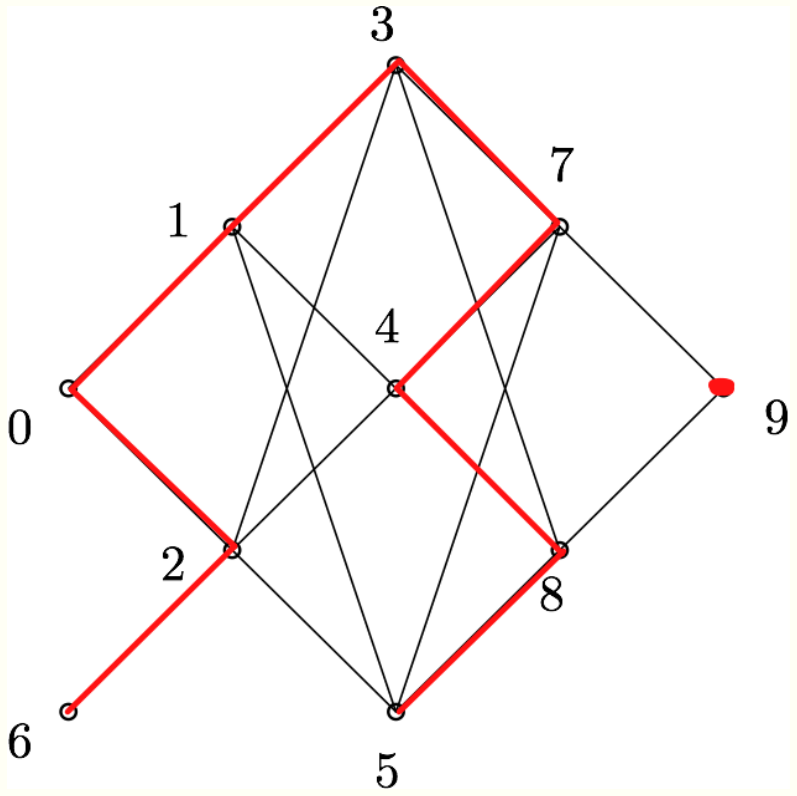
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering
2. $\tau = LDFS^{+}(\sigma) = 9, 8, 5, 7, 4, 1, 3, 2, 0, 6$

Let us take an example



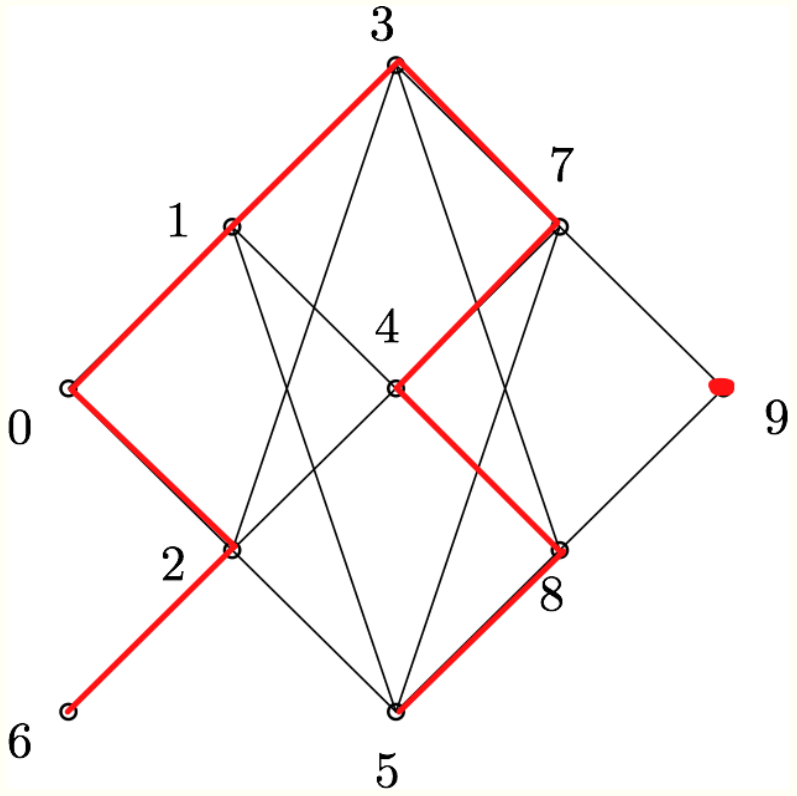
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering
2. $\tau = LDFS^{+}(\sigma) = 9, 8, 5, 7, 4, 1, 3, 2, 0, 6$
3. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Magic



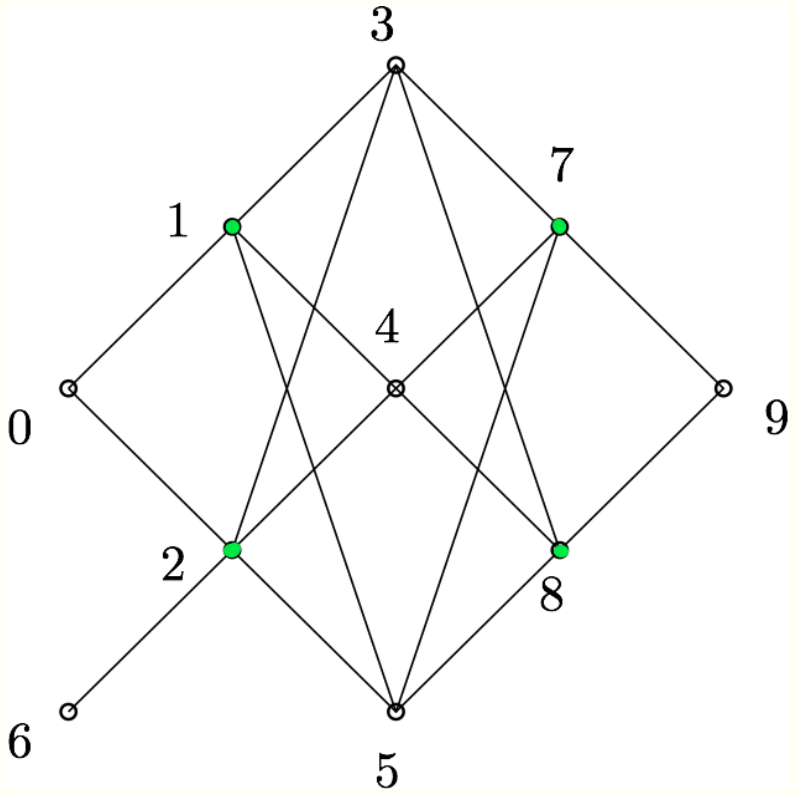
1. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Magic



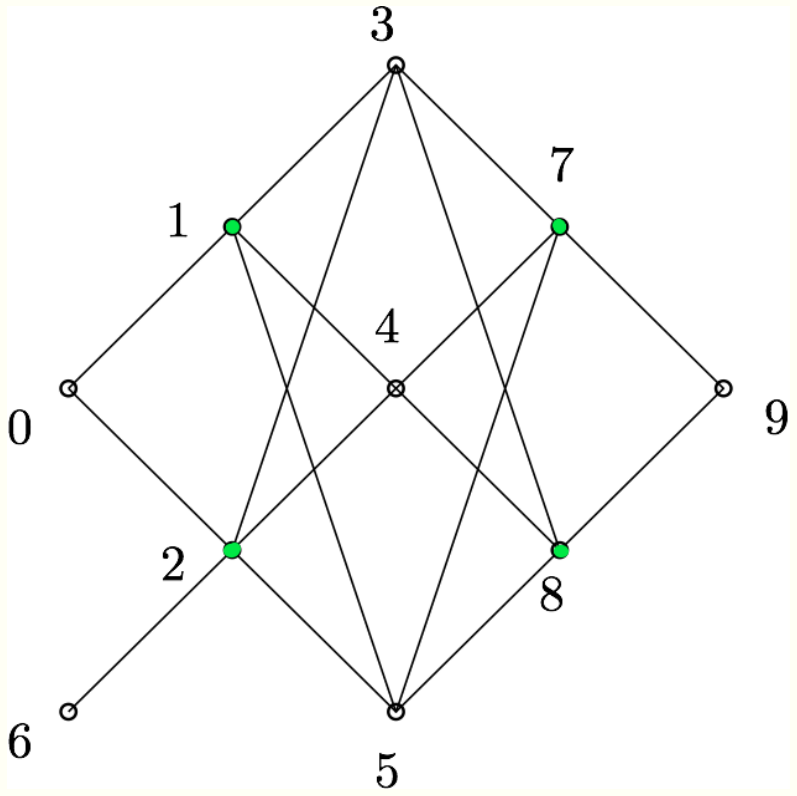
1. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$
2. The cutset $S = \{1, 7, 2, 8\}$ disconnects G into 6 connected components.

Cutset



1. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Cutset



1. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$
2. The cutset $S = \{1, 7, 2, 8\}$ disconnects G into 6 connected components.

Second Algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)

Second Algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.

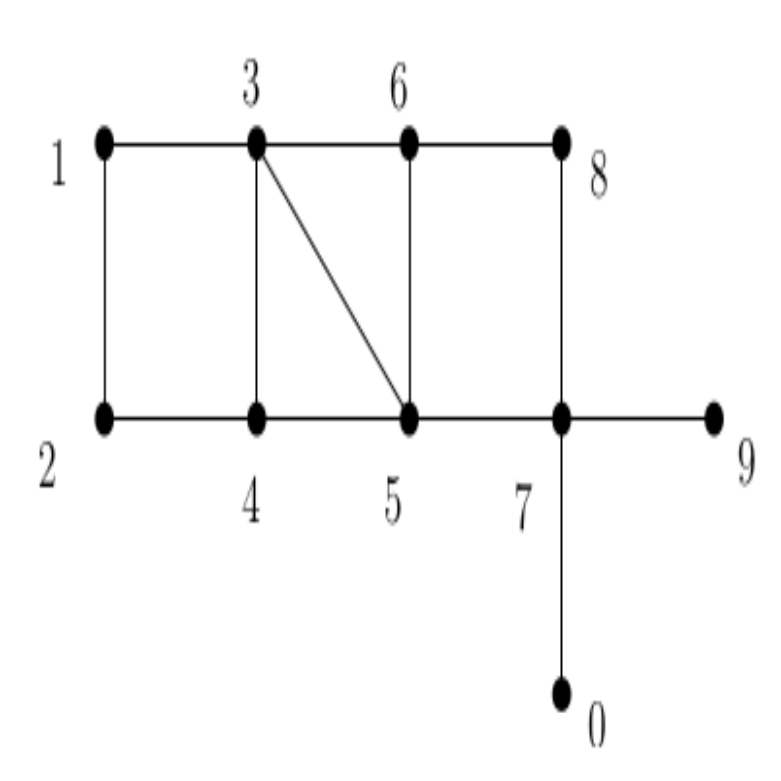
Second Algorithm

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.
3. Apply $GreedyIndependentSet(\tau)$ which gives the independent set

Second Algorithm

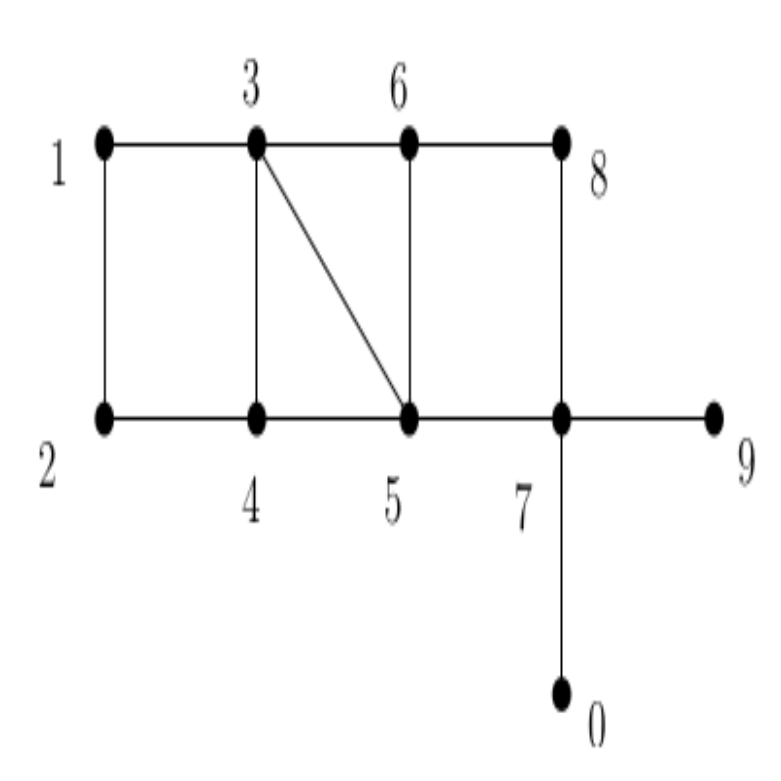
1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^{+}(\sigma)$ to produce an ordering τ.
3. Apply $GreedyIndependentSet(\tau)$ which gives the independent set
4. Exhibit a certificate of minimality with a cliques cover.

Let us take an example



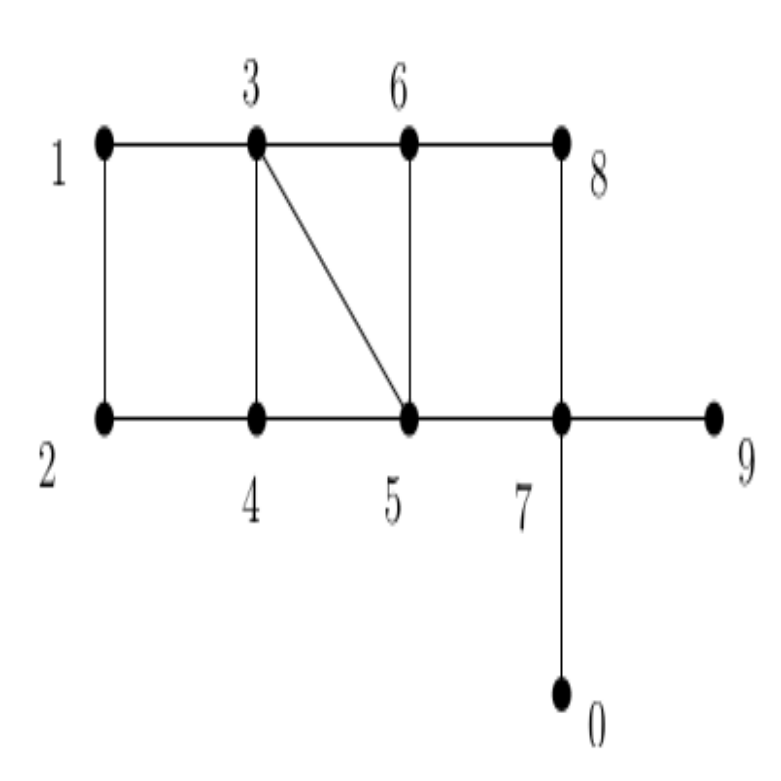
1. $\sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$ a co-comparability ordering

Let us take an example



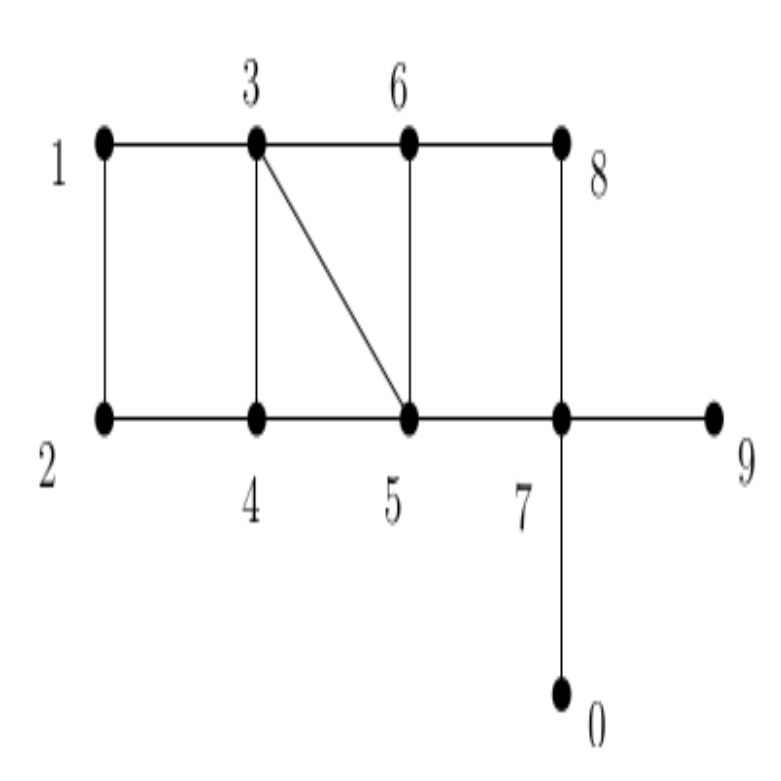
1. $\sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$ a co-comparability ordering
2. $\tau = LDFS^{+}(\sigma) = 0, 7, 9, 8, 6, 5, 3, 4, 2, 1$

Let us take an example



1. $\sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$ a co-comparability ordering
2. $\tau = LDFS^{+}(\sigma) = 0, 7, 9, 8, 6, 5, 3, 4, 2, 1$
3. $GreedyIndependentSet(\tau) = \{0, 9, 6, 4, 1\}$

Let us take an example



1. $\sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$ a co-comparability ordering
2. $\tau = LDFS^{+}(\sigma) = 0, 7, 9, 8, 6, 5, 3, 4, 2, 1$
3. $GreedyIndependentSet(\tau) = \{0, 9, 6, 4, 1\}$
4. Clique cover: $\{\{0\}, \{7, 9\}, \{8, 6\}, \{5, 3, 4\}, \{2, 1\}\}$

Greedy algorithm skeleton

1. Start with σ any co-comparability ordering of G (a linear extension of P)

Greedy algorithm skeleton

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^+(\sigma)$ to produce an ordering τ.

Greedy algorithm skeleton

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^+(\sigma)$ to produce an ordering τ.
3. Find the rightmost structure (ex: rightmost path cover, rightmost independent set ...) with respect to τ

Greedy algorithm skeleton

1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^+(\sigma)$ to produce an ordering τ.
3. Find the rightmost structure (ex: rightmost path cover, rightmost independent set ...) with respect to τ
4. Exhibit a certificate of optimality.

In other words

1. a preprocessing to obtain a cocomp ordering σ

In other words

1. a preprocessing to obtain a cocomp ordering σ
2. $LDFS^+(G, \sigma)$ produces a layered cocomp ordering τ.

In other words

1. a preprocessing to obtain a cocomp ordering σ
2. $LDFS^+(G, \sigma)$ produces a layered cocomp ordering τ.
3. Following τ collect in a greedy way an optimum solution

In other words

1. a preprocessing to obtain a cocomp ordering σ
2. $LDFS^+(G, \sigma)$ produces a layered cocomp ordering τ.
3. Following τ collect in a greedy way an optimum solution
4. Exhibit a certificate of optimality.

Complexity analysis

1. $O(n+m)$ to obtain a cocomp ordering σ. [1]

[1] Note that it is still not linear to check if σ is a cocomp ordering. Best known $O(n+m^{3/2})$ Michele Borassi, Raphael Gaudy, Michel Habib 2016.

Complexity analysis

1. $O(n+m)$ to obtain a cocomp ordering σ. [1]
2. $LDFS^+(G,\sigma)$ can be compute in $O(n+m)$ using a result from Lalla Mouatadib and Ekkehard Köler (SWAT 2014).

[1] Note that it is still not linear to check if σ is a cocomp ordering. Best known $O(n+m^{3/2})$ Michele Borassi, Raphael Gaudy, Michel Habib 2016.

Complexity analysis

1. $O(n+m)$ to obtain a cocomp ordering σ. [1]
2. $LDFS^+(G,\sigma)$ can be compute in $O(n+m)$ using a result from Lalla Mouatadib and Ekkehard Köler (SWAT 2014).
3. Following τ collect in a greedy way an optimum solution: need $O(n+m)$ in the two previous algorithms

[1] Note that it is still not linear to check if σ is a cocomp ordering. Best known $O(n+m^{3/2})$ Michele Borassi, Raphael Gaudy, Michel Habib 2016.

Complexity analysis

1. $O(n+m)$ to obtain a cocomp ordering σ. [1]
2. $LDFS^+(G,\sigma)$ can be compute in $O(n+m)$ using a result from Lalla Mouatadib and Ekkehard Köler (SWAT 2014).
3. Following τ collect in a greedy way an optimum solution: need $O(n+m)$ in the two previous algorithms
4. Exhibit a certificate of optimality: need $O(n+m)$ in the two previous algorithms

[1] Note that it is still not linear to check if σ is a cocomp ordering. Best known $O(n+m^{3/2})$ Michele Borassi, Raphael Gaudy, Michel Habib 2016.

Complexity analysis

1. $O(n+m)$ to obtain a cocomp ordering σ. [1]
2. $LDFS^+(G,\sigma)$ can be compute in $O(n+m)$ using a result from Lalla Mouatadib and Ekkehard Köler (SWAT 2014).
3. Following τ collect in a greedy way an optimum solution: need $O(n+m)$ in the two previous algorithms
4. Exhibit a certificate of optimality: need $O(n+m)$ in the two previous algorithms
5. So $O(n+m)$ in the whole.

[1] Note that it is still not linear to check if σ is a cocomp ordering. Best known $O(n+m^{3/2})$ Michele Borassi, Raphael Gaudy, Michel Habib 2016.

Permutation graphs

A graph G is a *permutation graph* if and only if G is the intersection graph of lines joining common entries of two permutations of $\{1, 2, \ldots |V|\}$ where the permutations are on parallel lines. The following characterization theorem is well known;

Theorem

An undirected graph G is a permutation graph if and only if G and its complement are comparability graphs; equivalently G and its complement are cocomparability graphs.

Recognition of Permutation graphs

PERMUTATION(G):

Input: A connected graph $G = (V, E)$, cocomp ordering σ of G and cocomp ordering τ of $\overline{G}$

Output: The message that at least one of σ, τ is not a cocomp ordering of its graph $(G, \overline{G})$ or total orderings π_1, π_2^{dual} of $1, 2, \cdots |V|$ that certify that G is a permutation graph

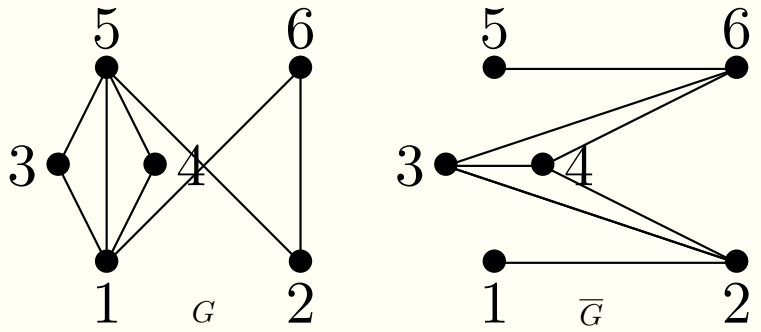
$P_{G(\tau)} \leftarrow$ an acyclic orientation of G using τ;

$\pi_1 \leftarrow dfgreedy^+(P_{G(\tau)}, \sigma)$;

$\pi_2 \leftarrow dfgreedy^+(P_{G(\tau)}, \sigma^{dual})$;

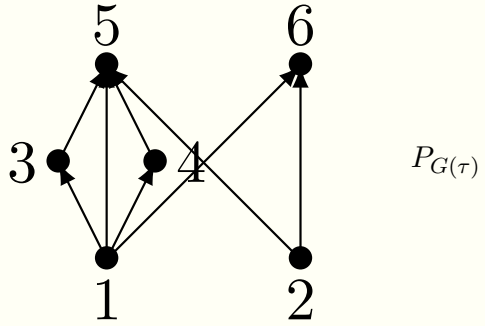
Check if π_1, π_2^{dual} represent G as a permutation graph;

If so, return (π_1, π_2^{dual}) else return - "at least one of σ, τ is not a cocomp ordering of its graph $(G, \overline{G})$."



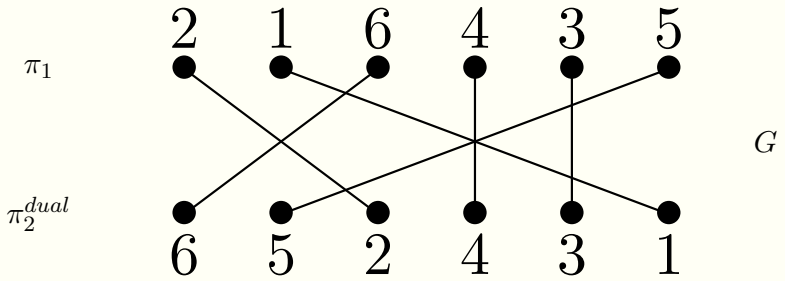
$\sigma = 1, 3, 5, 4, 2, 6$ is a cocomp ordering of G, but not a cocomp ordering of $\overline{G}$ (see umbrella (1, 5, 2))

$\tau = 1, 2, 3, 4, 5, 6$ is a cocomp ordering of $\overline{G}$, but not a cocomp ordering of G (see umbrella (2, 3, 6))



$$\pi_1 = 2, 1, 6, 4, 3, 5 = dfgreedy(P_{G(\tau)}, \sigma)$$

$$\pi_2 = 1, 3, 4, 2, 5, 6 = dfgreedy(P_{G(\tau)}, \sigma^{dual})$$



π_1 and π_2 are cocomp orderings of both G and $\overline{G}$

- dfgreedy is a graph search applied on posets and if σ is a cocomp then $dfgreedy^{+}(P_{G(\tau)}, \sigma)$ is also a cocomp.

- dfgreedy is a graph search applied on posets and if σ is a cocomp then $dfgreedy^{+}(P_{G(\tau)}, \sigma)$ is also a cocomp.
- If G is a permutation graph, $dfgreedy^{+}(P_{G(\tau)}, \sigma)$ and $dfgreedy^{+}(P_{G(\tau)}, \sigma^{dual})$ provide a representation of G as a permutation graph.

- So we have another example of this kind for the recognition of permutation graphs.

- So we have another example of this kind for the recognition of permutation graphs.
- The 2 first algorithms were known for interval graphs, and can be lifted to cocomparability graphs (with perhaps a new recent example from G. Mertzios?)

- So we have another example of this kind for the recognition of permutation graphs.
- The 2 first algorithms were known for interval graphs, and can be lifted to cocomparability graphs (with perhaps a new recent example from G. Mertzios?)
- Justifying my former questions:
  Why some of these algorithms based on graph searches look so greedy?
  What is the combinatorial structure involved?

Known fact about LDFS on cocomparability graphs

- produces an height preserving linear extension

Known fact about LDFS on cocomparability graphs

- produces an height preserving linear extension
- also yields a good arrangement of the vertices in a layer

The maximal antichain lattice of a poset (Berhendt 1988)

- From Birkhoff 1967, we know that the set of antichains of a poset yields a distributive lattice

The maximal antichain lattice of a poset (Berhendt 1988)

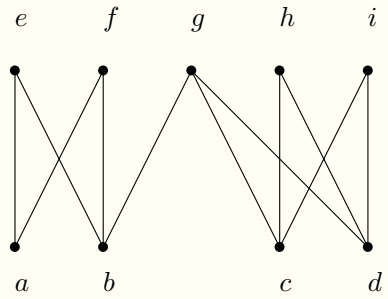
- From Birkhoff 1967, we know that the set of antichains of a poset yields a distributive lattice
- It is less known that the set of all maximal (resp. maximum sized from Dilworth) forms a lattice. We denoted by $\mathcal{MA}(P)$ this maximal antichain lattice for a partial order P.

The maximal antichain lattice of a poset (Berhendt 1988)

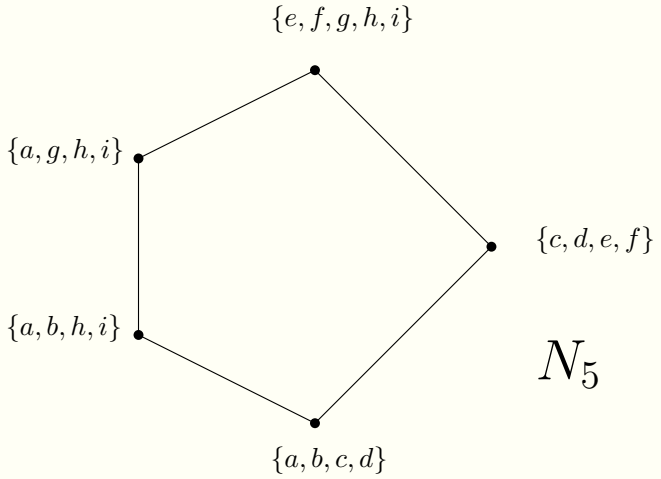
- From Birkhoff 1967, we know that the set of antichains of a poset yields a distributive lattice
- It is less known that the set of all maximal (resp. maximum sized from Dilworth) forms a lattice. We denoted by $\mathcal{MA}(P)$ this maximal antichain lattice for a partial order P.

Lemma

Let A, B be two maximal antichains of a partial order P.
$A \leq_{\mathcal{MA}(P)} B$ iff $\forall a \in A$, $\exists b \in B$ with $a \leq_P b$ iff $\forall b \in B$, $\exists a \in A$ with $a \leq_P b$.

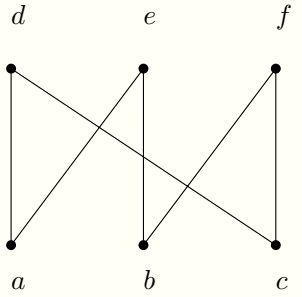


P_1

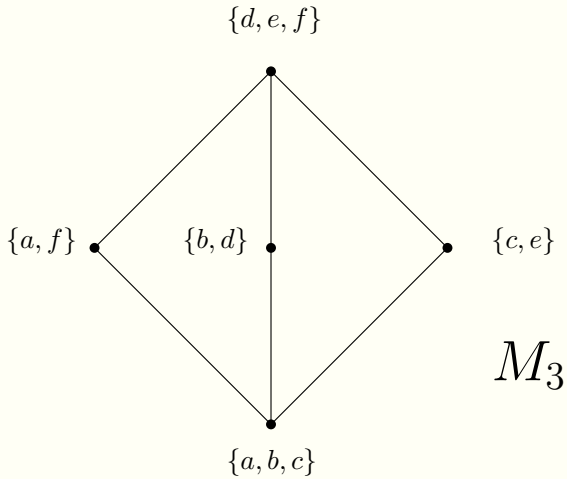


N_5

$MA(P_1)$



P_2



M_3

$MA(P_2)$

Figure: Two orders whose maximal antichain lattices are respectively N_5 and M_3, the smallest non distributive lattices.

Theorem

Berhendt (1988), Markowski (1975,1992): Any finite lattice is isomorphic to the lattice $\mathcal{MA}(P)$ of some finite partial order.

$\mathcal{MA}(P)$ can be exponential in the size of P

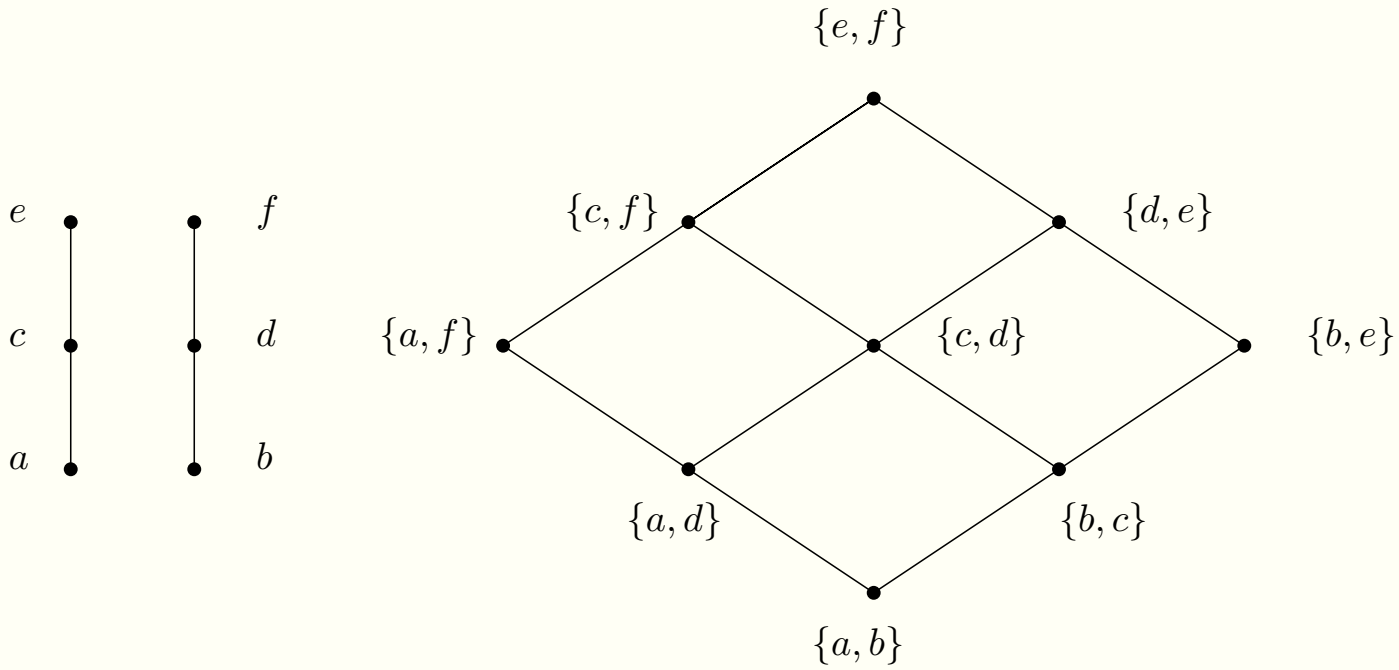


Figure: $\mathcal{MA}(P)$ for $k = 2$.

$\mathcal{MA}(P)$ not to be computed!
It is just a tool to understand the underlying structure.

Interval orders in this lattice

Theorem

P is an interval order if and only if $\mathcal{MA}(P)$ is a chain.

Interval orders in this lattice

Theorem

P is an interval order if and only if $\mathcal{MA}(P)$ is a chain.

Theorem

MH, Morvan, Rampon, Pouzet 1991: The minimal interval order extensions of P are in a one-to-one correspondence with the maximal chains of $\mathcal{MA}(P)$.

Interval orders in this lattice

Theorem

P is an interval order if and only if $\mathcal{MA}(P)$ is a chain.

Theorem

MH, Morvan, Rampon, Pouzet 1991: The minimal interval order extensions of P are in a one-to-one correspondence with the maximal chains of $\mathcal{MA}(P)$.

Consequences

The number of interval order extensions is a comparability invariant and #P-complete to compute.

Theorem

$G = (V(G), E(G))$ is a cocomparability graph if and only if $\mathcal{C}(G)$ can be equipped with a lattice structure $\mathcal{L}$ satisfying:

(i) *For every $A, B, C \in \mathcal{C}(G)$, such that $A \leq_{\mathcal{L}} B \leq_{\mathcal{L}} C$, then $A \cap C \subseteq B$.*

(ii) *For every $A, B \in \mathcal{C}(G)$, $(A \cup B) \subseteq (A \vee_{\mathcal{L}} B) \cup (A \wedge_{\mathcal{L}} B)$.*

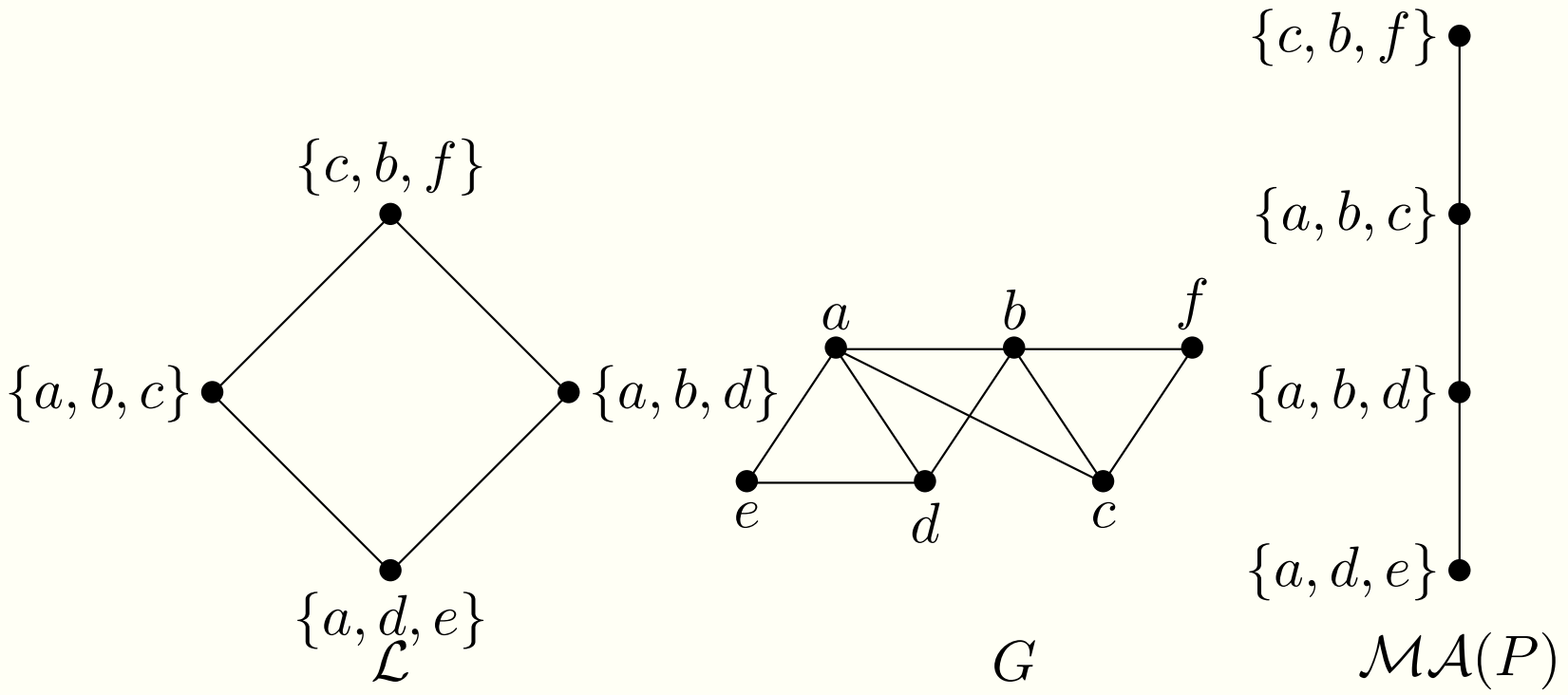


Figure: A graph G and a lattice $\mathcal{L}$ on $\mathcal{C}(G)$ that satisfies condition (i) and (ii) of Theorem. But $\mathcal{L}$ is not isomorphic to the lattice $\mathcal{MA}(P)$ for any partial order P that corresponds to a transitive orientation of $\overline{G}$. Since G is a prime interval graph, it has only one transitive orientation (up to reversal) which is an interval order and its maximal antichain lattice is a chain.

Theorem

Let G be a cocomparability graph and let $\mathcal{L}$ be a lattice structure on $\mathcal{C}(G)$ then $\mathcal{L}$ is isomorphic to a lattice $\mathcal{MA}(P)$ with P a transitive orientation of $\overline{G}$ if and only if the following conditions are satisfied:

(i) *For every $A, B, C \in \mathcal{C}(G)$, such that $A \leq_{\mathcal{L}} B \leq_{\mathcal{L}} C$, then $A \cap C \subseteq B$.*

(ii) *For every $A, B \in \mathcal{C}(G)$, $(A \cup B) \subseteq (A \vee_{\mathcal{L}} B) \cup (A \wedge_{\mathcal{L}} B)$.*

(iii) *For every $A, B \in \mathcal{C}(G)$, $(A \cap B) \subseteq (A \vee_{\mathcal{L}} B)$ and $(A \cap B) \subseteq (A \wedge_{\mathcal{L}} B)$.*

Corollary

For every lattice $\mathcal{L}$ associated to the maximal cliques of a cocomparability graph G and satisfying (i) and (ii) there exists a transitive orientation $R_{\mathcal{L}}$ of $\overline{G}$ such that $\mathcal{MA}(R_{\mathcal{L}})$ is an extension of $\mathcal{L}$.

Fulkerson and Gross 1965

Corollary

G is an interval graph if and only if $\mathcal{C}(G)$ can be equipped with a total order T satisfying for every C_i, C_j, C_k maximal cliques such that $C_i \leq_T C_j \leq_T C_k$, then $C_i \cap C_k \subseteq C_j$.

Corollary

Every maximal interval subgraph of a cocomparability graph is also a maximal chordal subgraph.

But

The converse is not true.

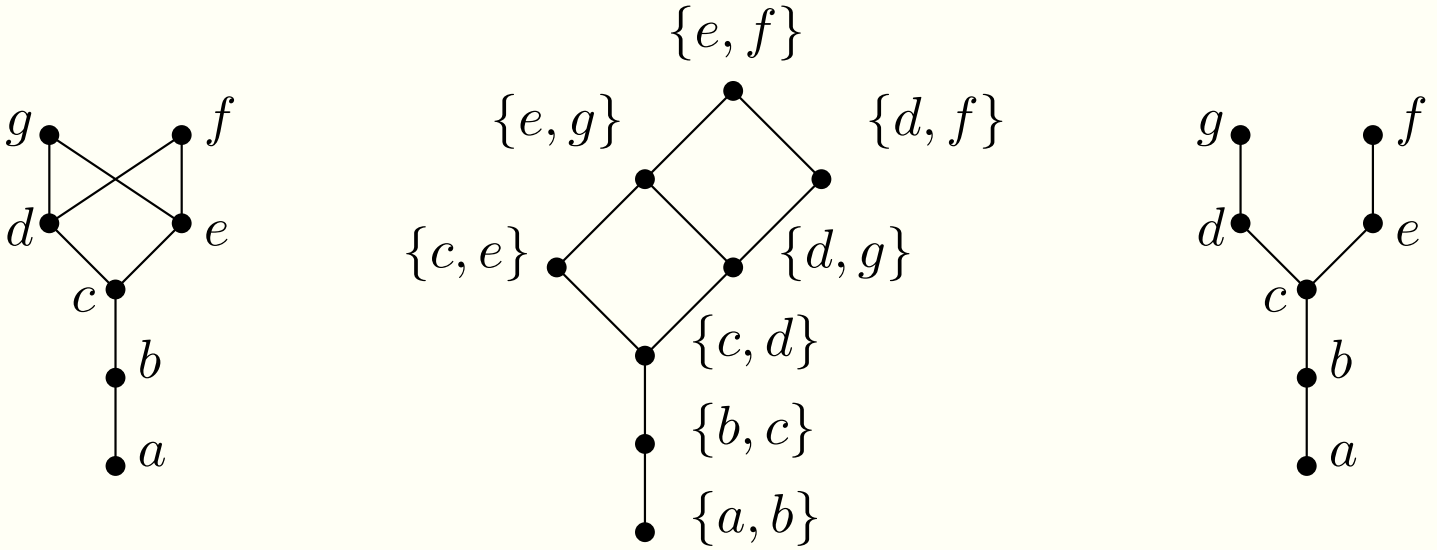


Figure: From left to right: a cocomparability graph, along with one of its lattices and a maximal chordal subgraph that is not an interval graph since it contains an asteroidal triple (a, f, g).

- We now describe an algorithm that computes an maximal interval subgraph of a cocomparability graph in $O(n + mlogn)$.

- We now describe an algorithm that computes an maximal interval subgraph of a cocomparability graph in $O(n + mlogn)$.
- Searching for an interval subgraph having maximum number of edges is NP-complete!

Chainclique Algorithm

Data: $G = (V(G), E(G))$ and a vertex ordering σ
Result: a chain of cliques C_1,...,C_j
$j \leftarrow 0$;
$i \leftarrow 1$;
$C_0 \leftarrow \emptyset$;
while $i \leq |V|$ **do**
  $j \leftarrow j + 1$ %{Starting a new clique}%;
  $C_j \leftarrow \{\sigma(i)\} \cup (N(\sigma(i)) \cap C_{j-1})$;
  $i \leftarrow i + 1$;
  while $i \leq |V|$ *and* $\sigma(i)$ *is universal to* C_j **do**
    $C_j \leftarrow C_j \cup \{\sigma(i)\}$ %{Augmenting the clique}%;
    $i \leftarrow i + 1$;
  end
end
Output $C_1, \ldots, C_j$;

Algorithm 1: Chainclique(G, σ)

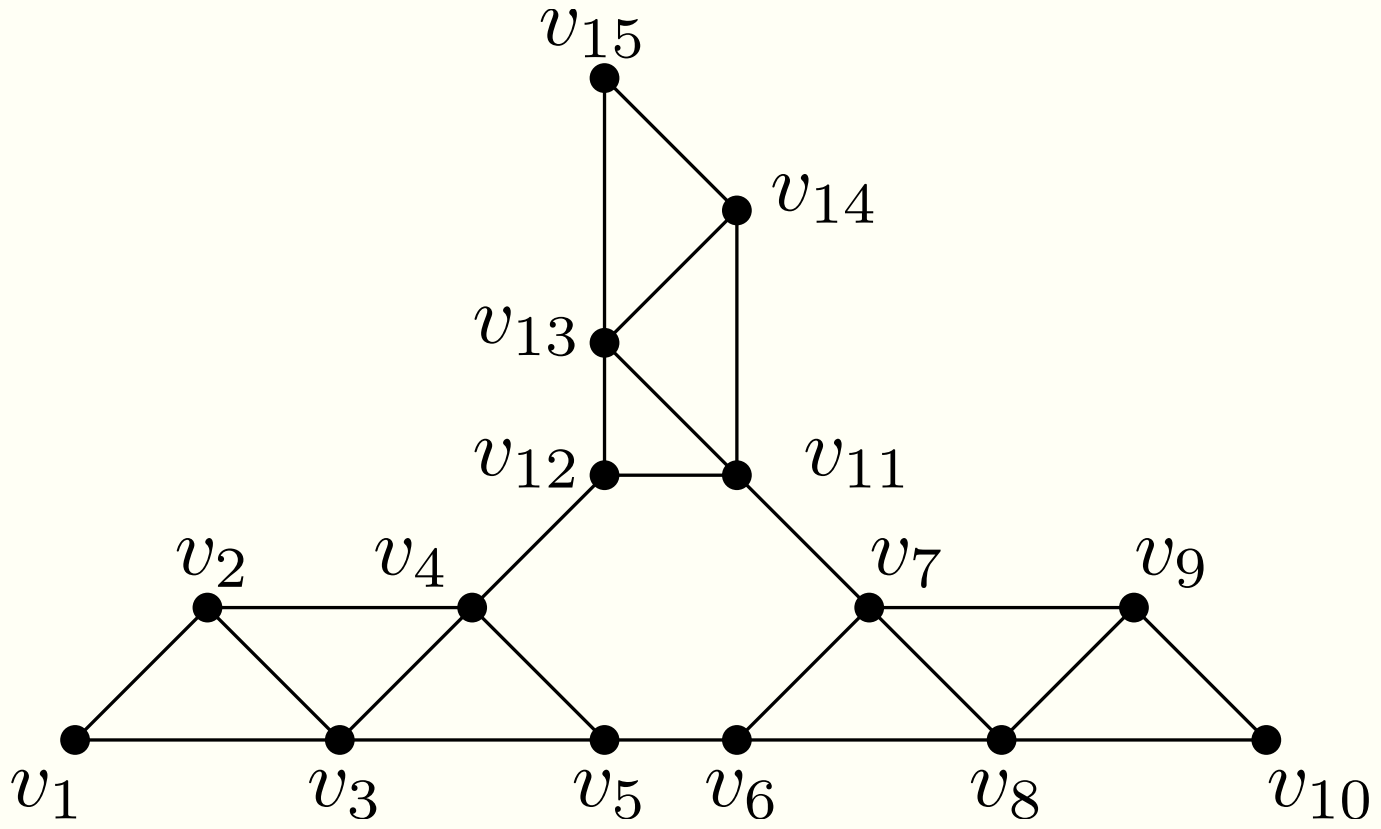


Figure: The graph $G = (V(G), E(G))$,
$\sigma = v_1,\ v_2,\ v_3,\ v_4,\ v_5,\ v_6,\ v_7,\ v_8,\ v_9,\ v_{10},\ v_{11},\ v_{12},\ v_{13},\ v_{14},\ v_{15}$.
Chainclique(G, σ) outputs:
$\{v_1,\ v_2,\ v_3\}$, $\{v_2,\ v_3,\ v_4\}$, $\{v_3,\ v_4,\ v_5\}$, $\{v_5,\ v_6\}$, $\{v_6,\ v_7,\ v_8\}$,
$\{v_7,\ v_8,\ v_9\}$, $\{v_8,\ v_9,\ v_{10}\}$, $\{v_{11},\ v_{12},\ v_{13}\}$, $\{v_{11},\ v_{13},\ v_{14}\}$,
$\{v_{13},\ v_{14},\ v_{15}\}$. $E(G) - E(\mathcal{C}_G) = \{\{v_7,\ v_{11}\},\ \{v_4,\ v_{12}\}\}$.

- In fact Chainclique greedily collects a chain of maximal cliques following σ

- In fact Chainclique greedily collects a chain of maximal cliques following σ
- Chainclique produces an interval subgraph which is maximal with respect to σ

- In fact Chainclique greedily collects a chain of maximal cliques following σ
- Chainclique produces an interval subgraph which is maximal with respect to σ
- Chaincliques can be done in linear time.

LocalMNS(G)

Data: $G = (V, E)$
Result: a total ordering σ such that $\sigma(i)$ is the i'th visited vertex
$D_1 \leftarrow \emptyset$;
$V' \leftarrow V$ $\%\{V'$ is the set of unchosen vertices$\}\%$;
$X \leftarrow \emptyset$ $\%\{X$ is the set of chosen vertices$\}\%$;
;
for $i = 1$ *to* $|V|$ **do**
  v is chosen as a vertex from V' with maximal neighborhood in D_i;
  $\sigma(i) \leftarrow v$;
  $V' \leftarrow V' - \{v\}$;
  $X \leftarrow X \cup \{v\}$;
  $D_{i+1} \leftarrow \{v\} \cup (N(v) \cap D_i)$; ;
  % Note: $x \in D_i$-$D_{i+1} \rightarrow x \notin D_j, j > i\%$;
end

1. LocalMNS is a graph search, kind of MNS, local because the next vertex to be visited has a maximal neighborhood on a strict subset of the visited vertices.

1. LocalMNS is a graph search, kind of MNS, local because the next vertex to be visited has a maximal neighborhood on a strict subset of the visited vertices.
2. LocalMNS can be implemented in linear time (just use a LocalMCS).

1. LocalMNS is a graph search, kind of MNS, local because the next vertex to be visited has a maximal neighborhood on a strict subset of the visited vertices.
2. LocalMNS can be implemented in linear time (just use a LocalMCS).
3. If τ is a cocomp ordering then also $\sigma = LocalMNS^{+}(G, \tau)$

Putting things together

1. Start with σ any co-comparability ordering of G. (a linear extension of P)

Putting things together

1. Start with σ any co-comparability ordering of G. (a linear extension of P)
2. Apply $LocalMNS^{+}(G, \sigma)$ to produce an ordering τ.

Putting things together

1. Start with σ any co-comparability ordering of G. (a linear extension of P)
2. Apply $LocalMNS^{+}(G, \sigma)$ to produce an ordering τ.
3. Find the rightmost chain of cliques (using Chaincliques) with respect to τ.

Putting things together

1. Start with σ any co-comparability ordering of G. (a linear extension of P)
2. Apply $LocalMNS^{+}(G, \sigma)$ to produce an ordering τ.
3. Find the rightmost chain of cliques (using Chaincliques) with respect to τ.
4. Exhibit a certificate of optimality.

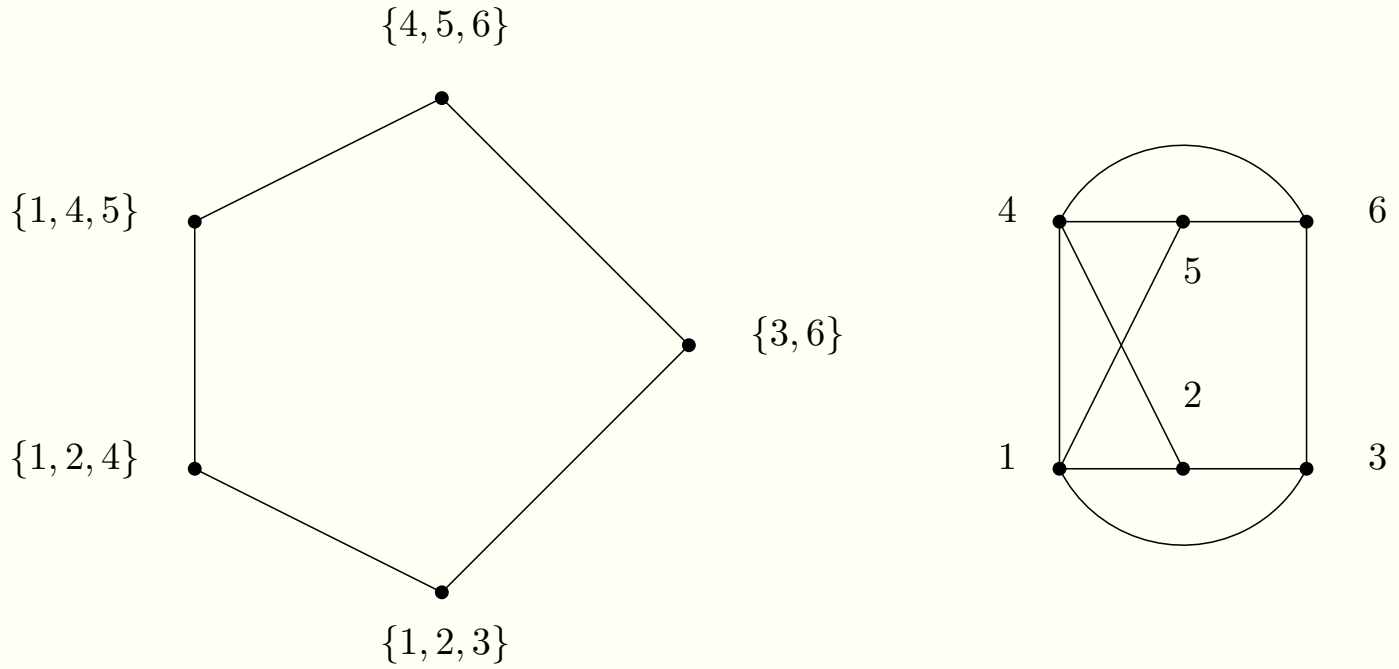


Figure: $\mathcal{MA}(P)$ and the corresponding cocomparability graph G. Let $\tau = 5, 6, 4, 2, 3, 1$ be a cocomp ordering.
$\sigma = LocalMNS^{+}(G, \tau) = 1, 3, 2, 4, 5, 6$ is a cocomp ordering and $Chainclique(G, \sigma)$ computes the maximal chain $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{1, 4, 5\}$ and $\{4, 5, 6\}$.

Theorem

$Chaincliques(G, \tau)$ produces a maximal chain of maximal cliques in $\mathcal{MA}(P_\tau)$ in $O(n + mlogn)$ time.
Therefore a maximal interval subgraph of G.

Theorem

$Chaincliques(G, \tau)$ produces a maximal chain of maximal cliques in $\mathcal{MA}(P_\tau)$ in $O(n + mlogn)$ time.
Therefore a maximal interval subgraph of G.

Theorem

For a cocomparability graph G and a transitive orientation P of $\overline{G}$, every maximal chain of $\mathcal{MA}(P)$ can be computed using $Chainclique(G, \sigma)$ on some cocomp LocalMNS ordering σ.

Computing all simplicial vertices in linear time

1. Start with σ any co-comparability ordering of G. (a linear extension of P)

Computing all simplicial vertices in linear time

1. Start with σ any co-comparability ordering of G. (a linear extension of P)
2. Apply $MNS^+(G, \sigma)$ to produce an ordering τ.

Computing all simplicial vertices in linear time

1. Start with σ any co-comparability ordering of G. (a linear extension of P)
2. Apply $MNS^+(G, \sigma)$ to produce an ordering τ.
3. Find the rightmost chain of cliques (using Chaincliques) with respect to τ.

Computing all simplicial vertices in linear time

1. Start with σ any co-comparability ordering of G. (a linear extension of P)
2. Apply $MNS^+(G, \sigma)$ to produce an ordering τ.
3. Find the rightmost chain of cliques (using Chaincliques) with respect to τ.
4. Collect greedily the simplicial vertices following τ.

- So far we have 2 more examples of our algorithm skeleton (changing LDFS into LocalMNS or MNS).

- So far we have 2 more examples of our algorithm skeleton (changing LDFS into LocalMNS or MNS).
- Can we explain the greediness?

Other algorithmic application of this lattice structure

1. A linear time algorithm to compute a minimal clique separator decomposition of a cocomparability graph. Which has the same flavor that the previous algorithm for computing all simplicial vertices.

Lexicographic Breadth-First Search

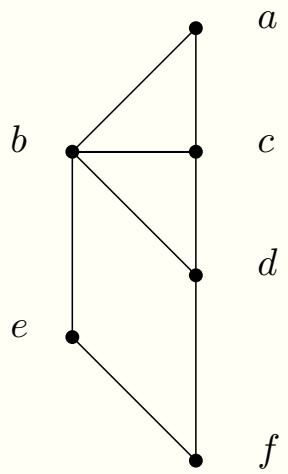
LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



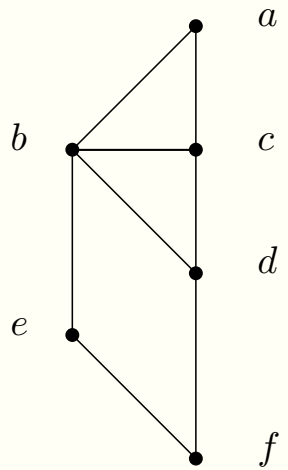
vertex	label
d	ϵ
c	ϵ
b	ϵ
f	ϵ
a	ϵ
e	ϵ

$i = 1,\ \sigma =$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



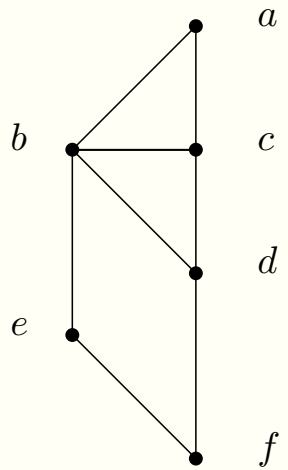
vertex	label
c	ϵ
b	ϵ
f	ϵ
a	ϵ
e	ϵ

$i = 1,\ \sigma = d$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



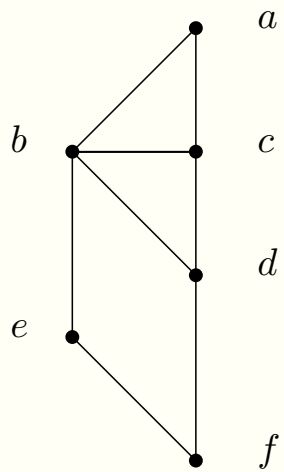
vertex	label
c	5
b	5
f	5
a	ϵ
e	ϵ

$i = 2,\ \sigma = d$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



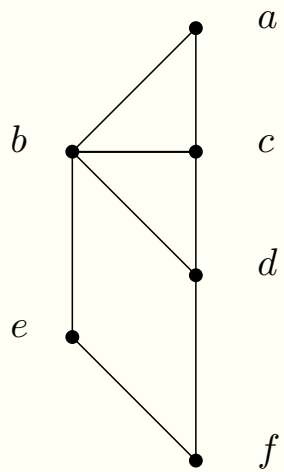
vertex	label
b	5
f	5
a	ϵ
e	ϵ

$i = 2,\ \sigma = d,\ c$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



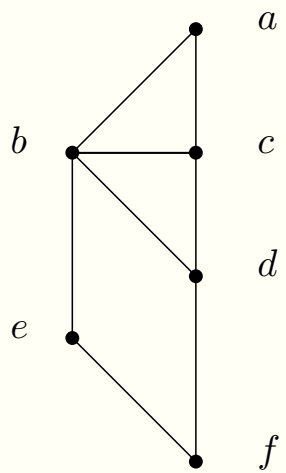
vertex	label
b	54
f	5
a	4
e	ϵ

$i = 3,\ \sigma = d,\ c$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



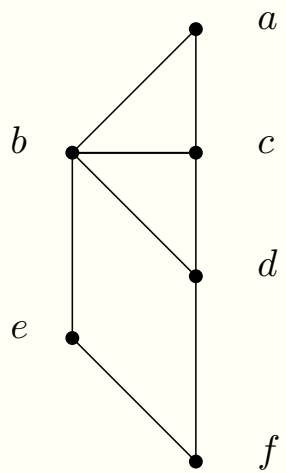
vertex	label
f	5
a	4
e	ϵ

$i = 3,\ \sigma = d,\ c,\ b$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



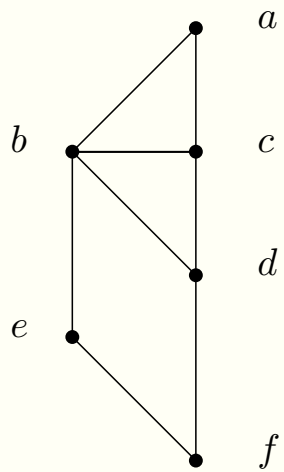
vertex	label
f	5
a	43
e	3

$i = 4,\ \sigma = d,\ c,\ b$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



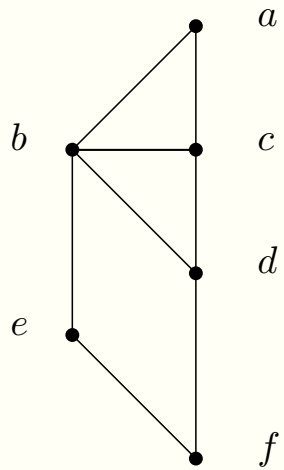
vertex	label
a	43
e	3

$i = 4,\ \sigma = d,\ c,\ b,\ f$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



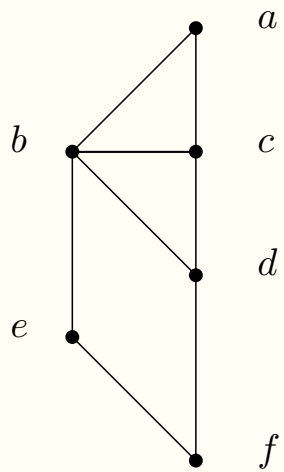
vertex	label
a	43
e	32

$i = 5,\ \sigma = d,\ c,\ b,\ f$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



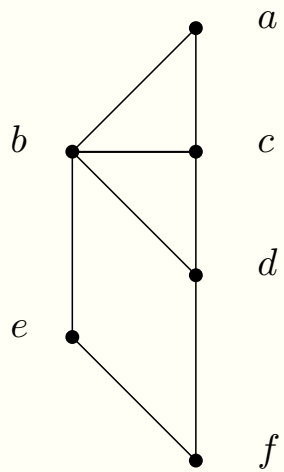
vertex	label
e	32

$i = 5,\ \sigma = d,\ c,\ b,\ f,\ a$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



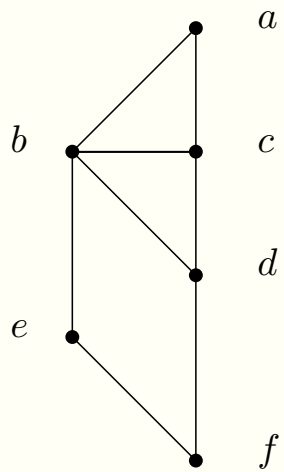
vertex	label
e	32

$i = 6,\ \sigma = d,\ c,\ b,\ f,\ a$

Lexicographic Breadth-First Search

LBFS

Input: an undirected graph $G = (V(G), E(G))$;
Assign label ϵ to all vertices
For $(i = 1$ **to** $|V(G)|)$ **do**
  Visit v, any unvisited vertex with lexicographically largest label
  For each unvisited vertex $u \in N(v)$ **do**
    Append letter $|V(G)| - i$ to $label(u)$



vertex	label

$i = 6,\ \sigma = d,\ c,\ b,\ f,\ a,\ e$

LBFS is very powerful as a preprocessing in a series of algorithms playing with hereditary classes of graphs, but not only:

- Chordal graphs, cographs, proper interval graphs, interval graphs, circle graphs, ...

LBFS is very powerful as a preprocessing in a series of algorithms playing with hereditary classes of graphs, but not only:

- Chordal graphs, cographs, proper interval graphs, interval graphs, circle graphs, ...
- To design linear time algorithms for modular decomposition, split decomposition

LBFS is very powerful as a preprocessing in a series of algorithms playing with hereditary classes of graphs, but not only:

- Chordal graphs, cographs, proper interval graphs, interval graphs, circle graphs, ...
- To design linear time algorithms for modular decomposition, split decomposition
- ...

Repeated LBFS$^+$

Repeated LBFS$^+$

Input: $G = (V(G), E(G))$ an undirected graph;
$\sigma_1 \leftarrow$ LBFS(G);
For $i = 2$ **to** $|V(G)|$ **do**
  $\sigma_i \leftarrow$ LBFS$^+$(G,σ_{i-1});
Output $\sigma_{|V(G)|}$;

Repeated LBFS$^+$

Repeated LBFS$^+$

Input: $G = (V(G), E(G))$ an undirected graph;
$\sigma_1 \leftarrow$ LBFS(G);
For $i = 2$ **to** $|V(G)|$ **do**
  $\sigma_i \leftarrow$ LBFS$^+$(G,σ_{i-1});
Output $\sigma_{|V(G)|}$;

Theorem

$G = (V(G), E(G))$ is a cocomparability graph if and only if the Repeated LBFS$^+$ algorithm computes a cocomp ordering.

Repeated LBFS$^+$

Best possible

Using a Ma's family of interval graphs (2000), this result is best possible, i.e., a constant number of LBFS would not be enough for all graphs.

remark

The same algorithm produces an interval ordering on interval graph.

Conjecture:

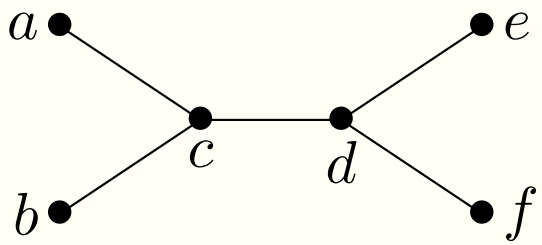
For a cocomp graph, this series of LBFS$^+$ orderings always ends in a 2-cycle (kind of fixed point).

Potential Algorithm, it relies on a positive answer to the conjecture

Input: G a connected graph
Output: a cocomp ordering of G iff G is a cocomparability graph

$\sigma_1 \leftarrow \text{LBFS}(G)$;
$\sigma_2 \leftarrow \text{LBFS}^+(G, \sigma_1)$;
$\sigma_3 \leftarrow \text{LBFS}^+(G, \sigma_2)$;
$i \leftarrow 3$;
while $\sigma_i \neq \sigma_{i-2}$ **do**
  $i \leftarrow i + 1$;
  $\sigma_i \leftarrow \text{LBFS}^+(G, \sigma_{i-1})$;
end
Output σ_i;

Algorithm 2: Cocomp ordering potential Algorithm



$$\sigma = LBFS(G, c) = c, d, a, b, e, f$$
$$\tau = LBFS^{+}(G, \sigma) = f, d, e, c, b, a$$
$$\theta = LBFS^{+}(G, \tau) = a, c, b, d, e, f$$
$$\tau = LBFS^{+}(G, \theta) = f, d, e, c, b, a$$

: 2-cycle between θ and τ
but $\theta \neq \tau^{d}$.

Figure: A non proper interval graph

- If the conjecture is true, the above algorithm also provides a certificate than can be checked in linear time.

- If the conjecture is true, the above algorithm also provides a certificate than can be checked in linear time.
- With Lalla Mouatadid, Pierre Charbit, Reza Naserasr, we proved the conjecture for trees, proper interval graphs, interval graphs, cobipartite graphs, and some other particular cases.

Conclusions

- Can we compute LocalMNS$^+(G, \sigma)$ in linear time ?

[2] An asteroidal triple or AT, is triple of non adjacent vertices such that for every two of them there exists a path joigning them avoiding the neighborhood of the third.

Conclusions

- Can we compute LocalMNS$^+(G, \sigma)$ in linear time ?
- Study the maximal clique structure of AT-free[2] graphs, since: they generalize cocomparability graphs

[2] An asteroidal triple or AT, is triple of non adjacent vertices such that for every two of them there exists a path joigning them avoiding the neighborhood of the third.

Conclusions

- Can we compute LocalMNS$^+(G, \sigma)$ in linear time ?
- Study the maximal clique structure of AT-free[2] graphs, since: they generalize cocomparability graphs
- It is well-known that:
  G AT-free iff G^2 cocomparability.

[2] An asteroidal triple or AT, is triple of non adjacent vertices such that for every two of them there exists a path joigning them avoiding the neighborhood of the third.

Many thanks for your attention!

Muchas gracias por su atención!

Muito obrigado pela atenç$\breve{a}$o!

Trugarez ha ken ar wech all!

Merci