New perspectives of graph searches on structured families of graphs

Michel Habib
habib@irif.fr
http://www.irif.fr/~habib

CIRM : Alea 2018 14 mars 2018

Schedule

Motivation to the study of graph searches

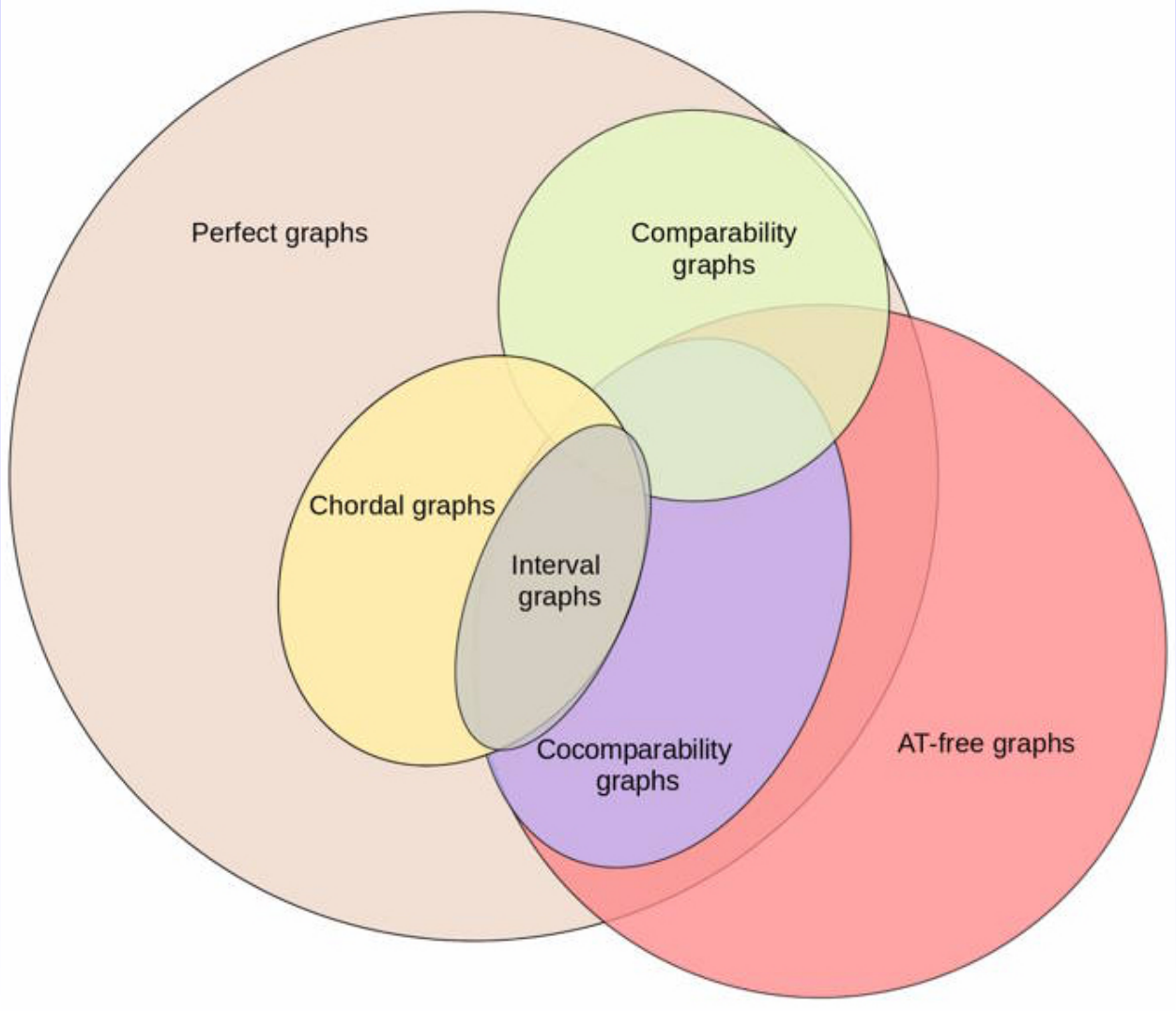
Diameter computations with BFS

A cycling problem with LexBFS

Greedy aspects of LDFS on cocomparability graphs

Conclusions and perspectives

FIGURE – Some well-structured graph classes



- We are really involved in efficient graph algorithm design, trying to understand why greedy algorithms work so well in practice.

- We are really involved in efficient graph algorithm design, trying to understand why greedy algorithms work so well in practice.
- We mostly play with linear time algorithms or heuristics.

- We are really involved in efficient graph algorithm design, trying to understand why greedy algorithms work so well in practice.
- We mostly play with linear time algorithms or heuristics.
- But our algorithms are always based on graph structures (even if we do not always understand why !)

Some definitions

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.

Graph searches are very well known and used in many situations :

1. "Fil d'ariane" in the Greek mythology.
2. Euler (1735) for solving the famous walk problem in Kœnisberg city
3. Euler's theorem proved by Hierholzer in 1873.
4. Tremaux (1882) and Tarry (1895) using DFS to solve maze problems
5. Fleury, proposed a nice algorithm to compute an Euler Tour, cited in E. Lucas, Récréations mathématiques, Paris, 1891.
6. Computer scientists from 1950, in particular in the 70's, Tarjan for new applications of DFS....
7. 4 points characterizations Corneil, Krueger (2008), and the definition of LDFS a new interesting basic search.

LE PROBLÈME DES LABYRINTHES;

PAR M. G. TARRY.

Tout labyrinthe peut être parcouru en une seule course, en passant deux fois en sens contraire par chacune des allées, sans qu'il soit nécessaire d'en connaître le plan.

Pour résoudre ce problème, il suffit d'observer cette règle unique :

Ne reprendre l'allée initiale qui a conduit à un carrefour pour la première fois que lorsqu'on ne peut pas faire autrement.

Nous ferons d'abord quelques remarques.

A un moment quelconque, avant d'arriver à un car-

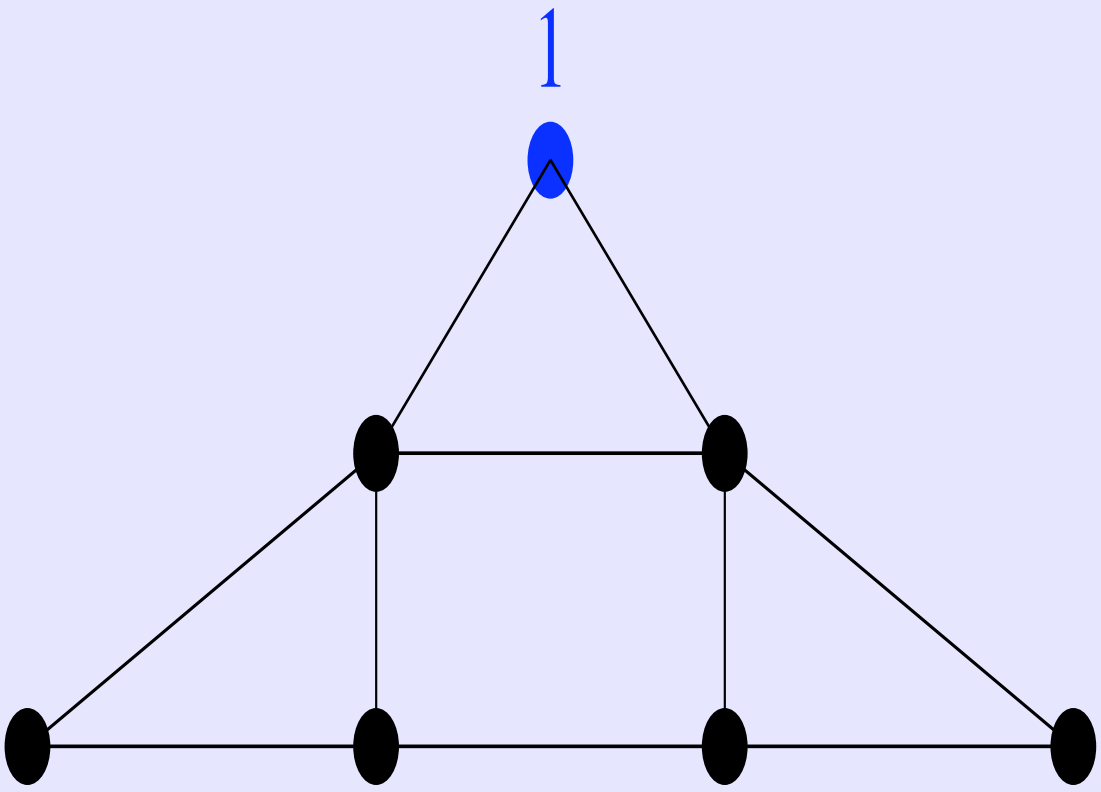
G. Tarry, *Le problème des labyrinthes*, **Nouvelles Annales de Mathématique**, Vol 14 : 187-190 (1895).

They are many formalisms to represent graph search depending on the view point :

- Data structures involved
- Implementation issues
- Path algebras
- Bio-inspired graph searches (as used by Amos Korman with ants)
- Reachability problems in complexity theory
- Here we want to focus on the visiting ordering of the vertices and on the tie-break process that distinguishes graph searches

1. For us a graph search just produces a total ordering of the vertices. Can be seen as a streaming algorithm visiting the graph leaving just a number in each node a label with *logn* bits.
2. In the following an ordering of the vertices, always means a total ordering of the vertices.
3. Seminal paper with a systematic study of graph search : D.G. Corneil et R. M. Krueger, A unified view of graph searching, SIAM J. Discrete Math, 22, Num 4 (2008) 1259-1276

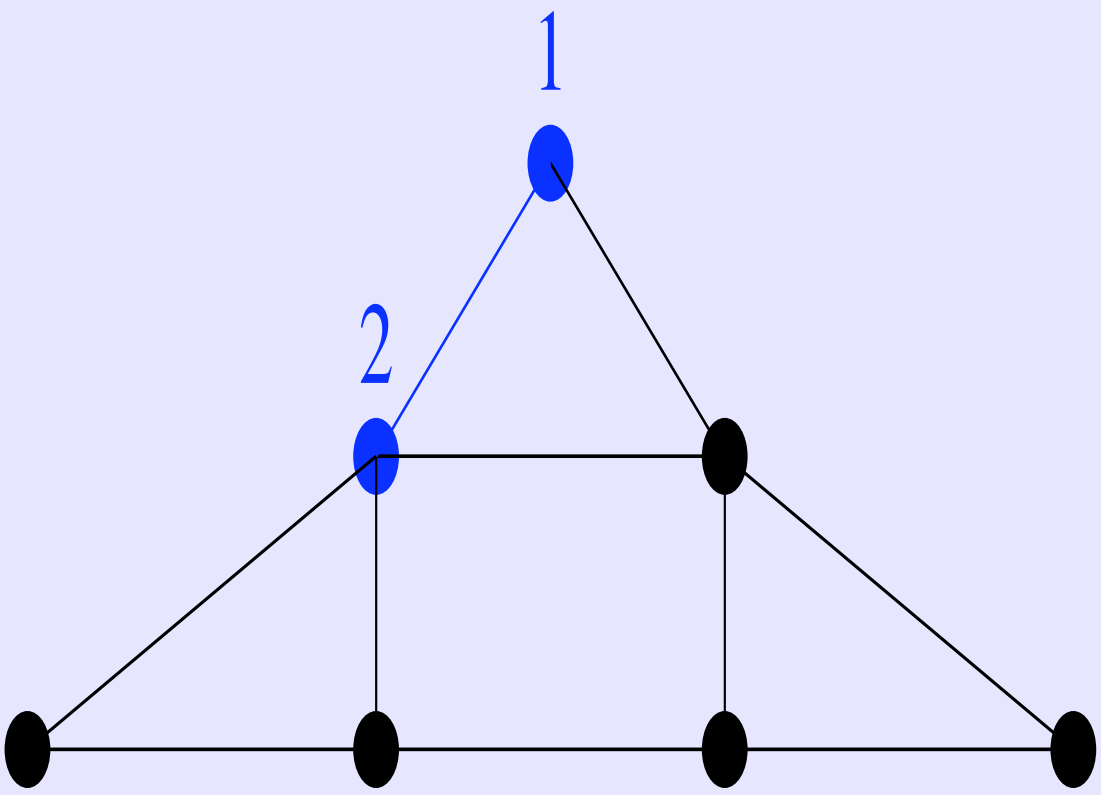
Generic Search



Invariant

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

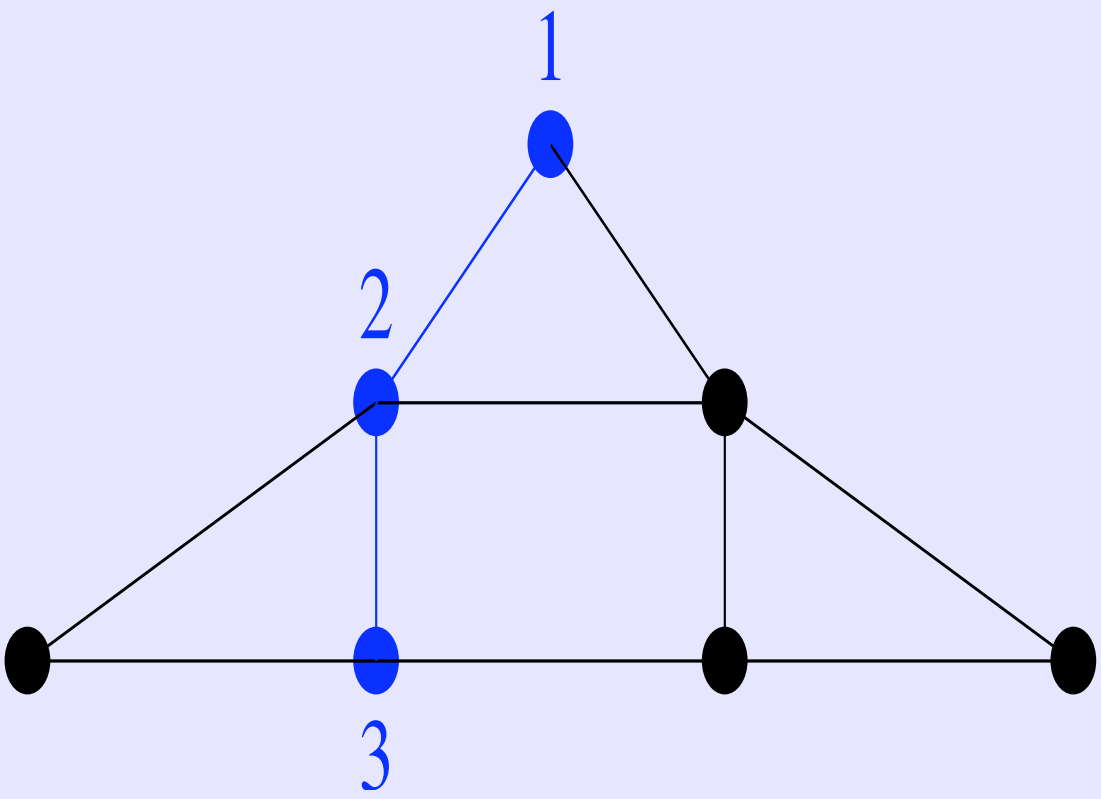
Generic Search



Invariant

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

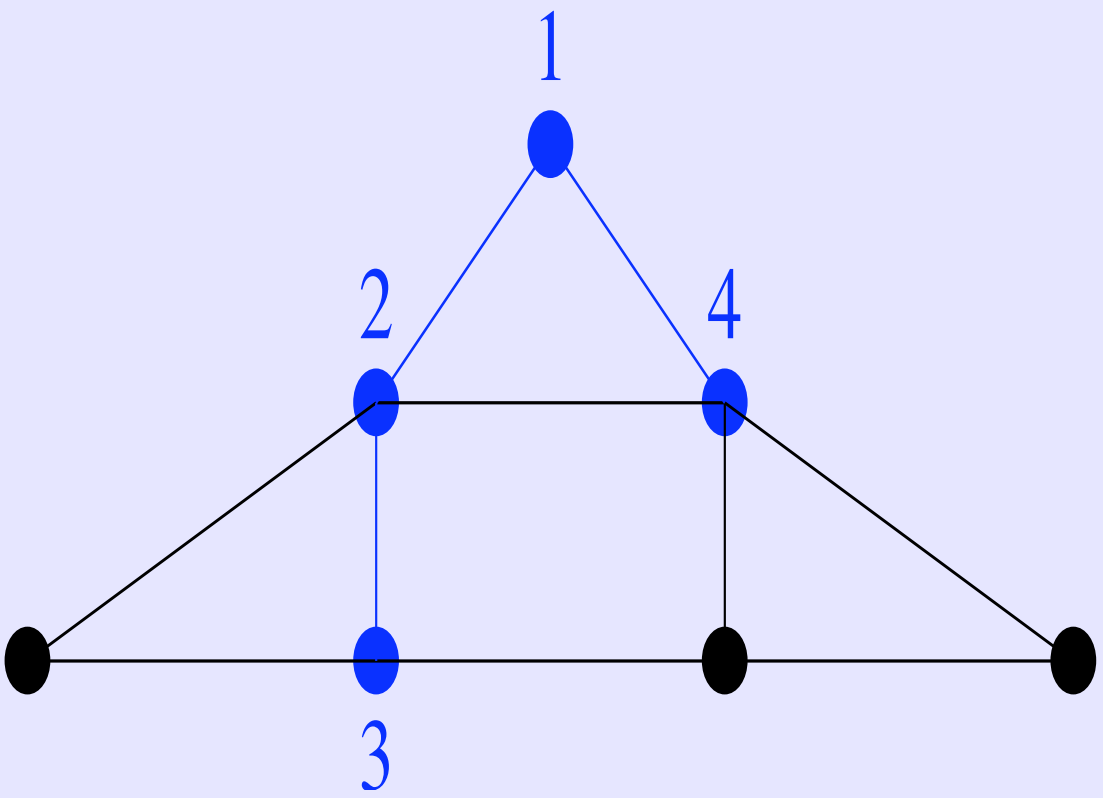
Generic Search



Invariant

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

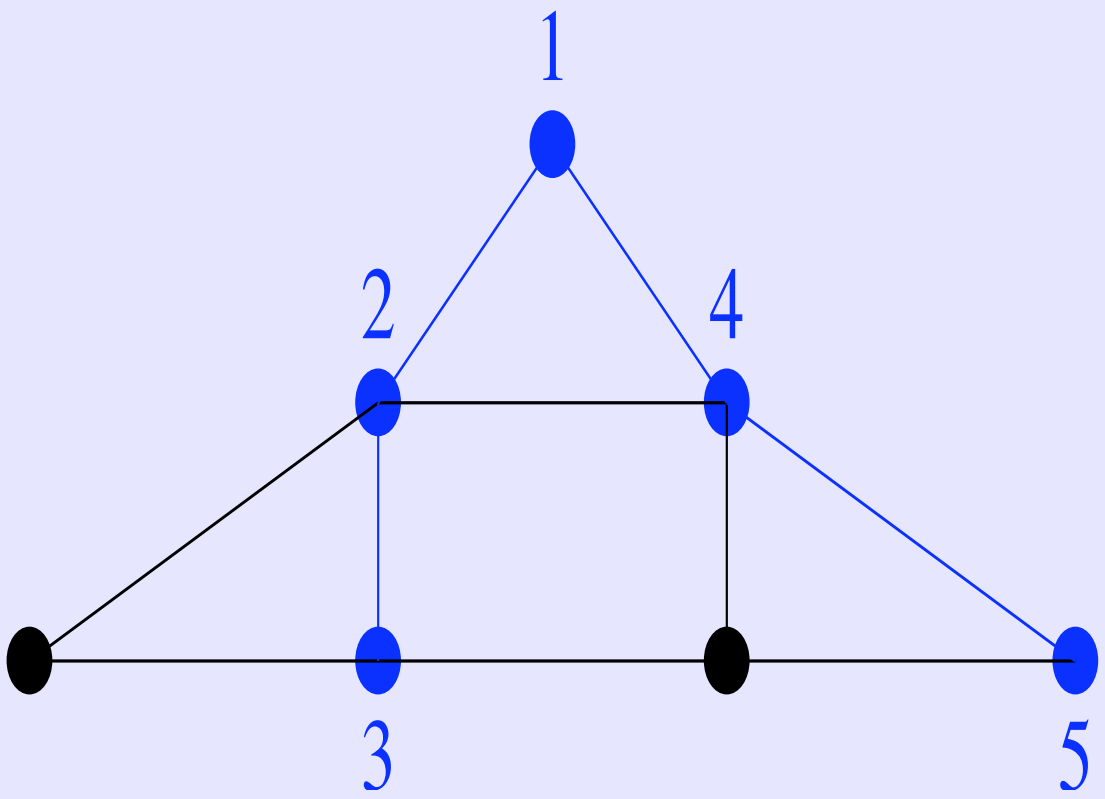
Generic Search



Invariant

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

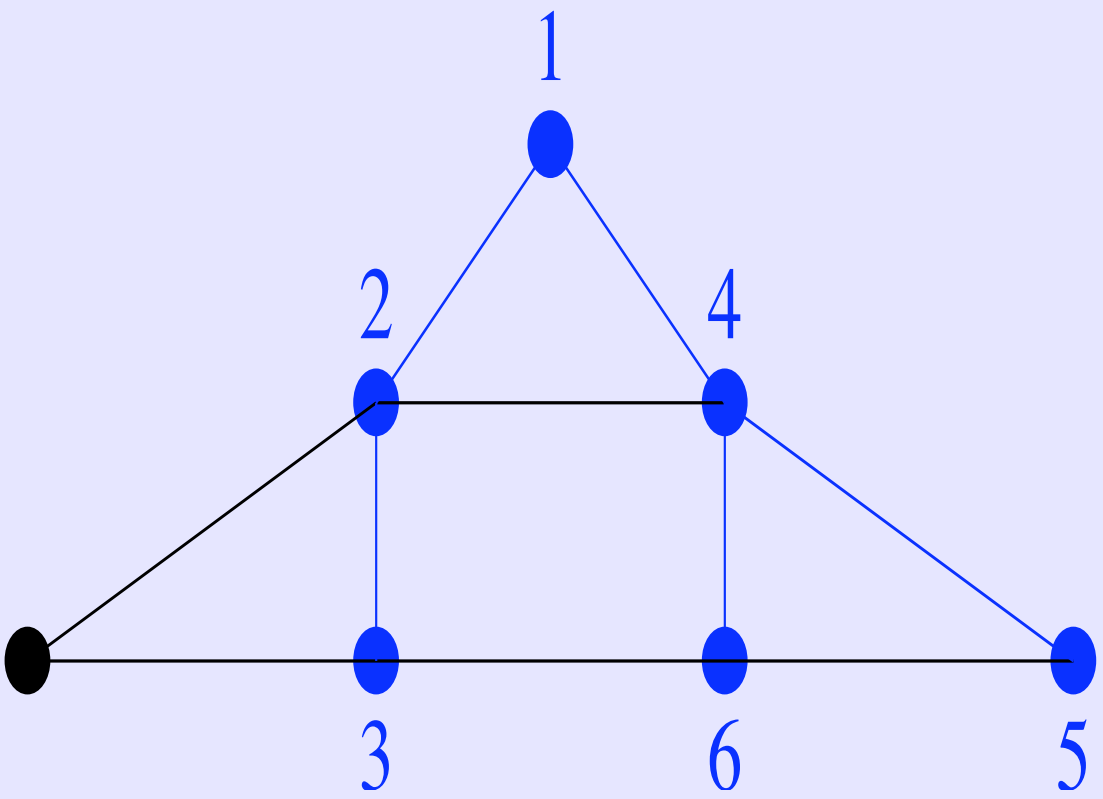
Generic Search



Invariant

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

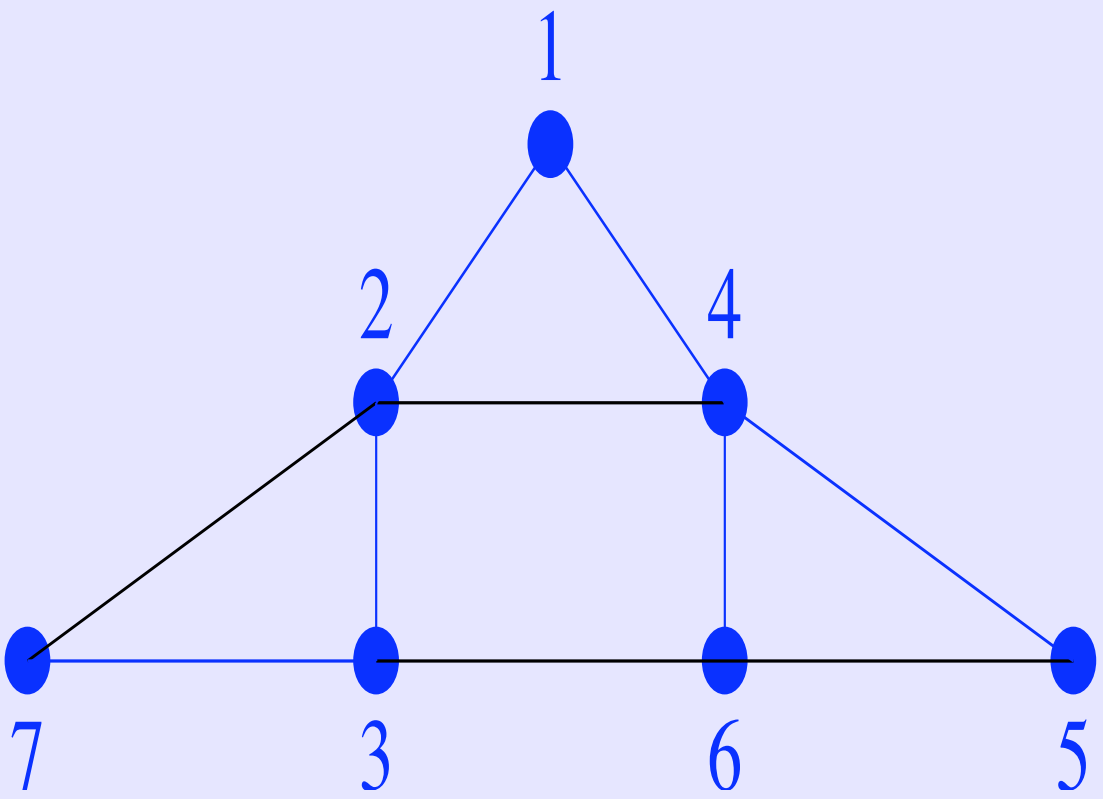
Generic Search



Invariant

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

Generic Search



Invariant

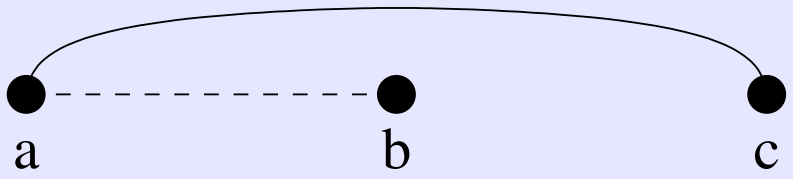
At each step, an edge between a visited vertex and a unvisited one is selected

Generic search

$S \leftarrow \{s\}$
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

Generic question ?

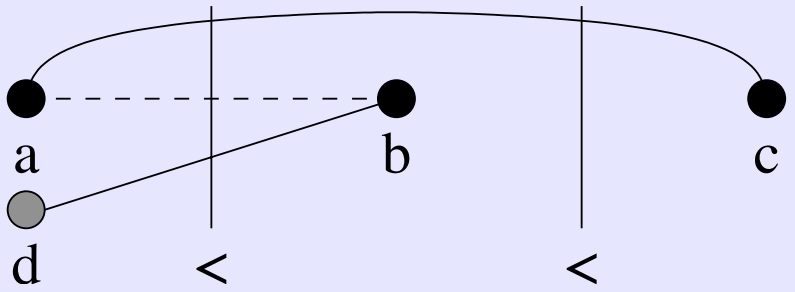
Let a, b et c be 3 vertices such that $ab \notin E$ et $ac \in E$.



Under which condition could we visit first a then b and last c ?

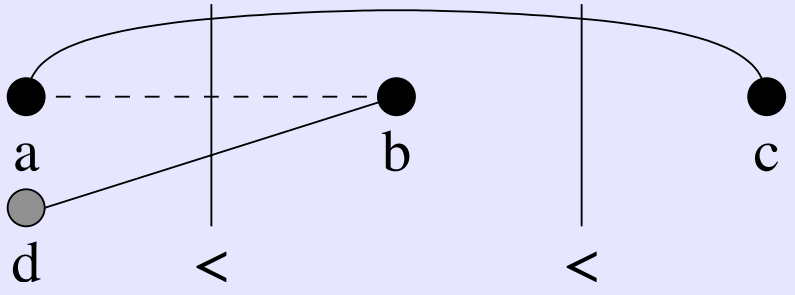
Property (Generic)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma b$ et $db \in E$



Property (Generic)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma b$ et $db \in E$



Theorem

For a graph $G = (V, E)$, an ordering σ on V is a generic search of G iff σ satisfies property (Generic).

Most of the searches that we will study are refinement of this generic search

- i.e. we just add new rules to follow for the choice of the next vertex to be visited

Most of the searches that we will study are refinement of this generic search

- i.e. we just add new rules to follow for the choice of the next vertex to be visited
- DFS (Stack), BFS(Queue), Dijkstra (Heap), ...

Most of the searches that we will study are refinement of this generic search

- i.e. we just add new rules to follow for the choice of the next vertex to be visited
- DFS (Stack), BFS(Queue), Dijkstra (Heap), ...
- Graph searches mainly differ by the management of the tie-break set

BFS

Algorithm 1: Breadth First Search (BFS)

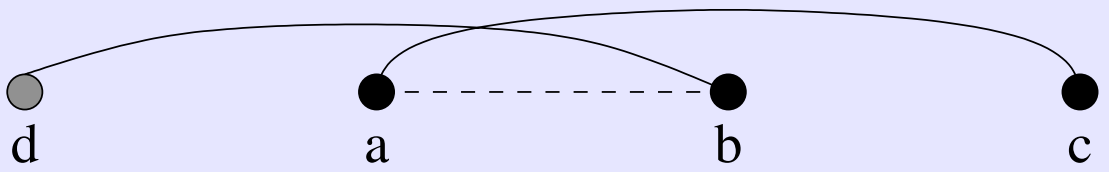
Data: a graph $G = (V, E)$ and a start vertex $s \in V$

Result: an ordering σ of V

Initialize *queue* to $\{s\}$
for $i \leftarrow 1$ **à** n **do**
    dequeue v from beginning of *queue*
    $\sigma(i) \leftarrow v$
    foreach *unnumbered vertex* $w \in N(v)$ **do**
        if w *is not already in queue* **then**
            enqueue w to the end of *queue*
        end
    end
end

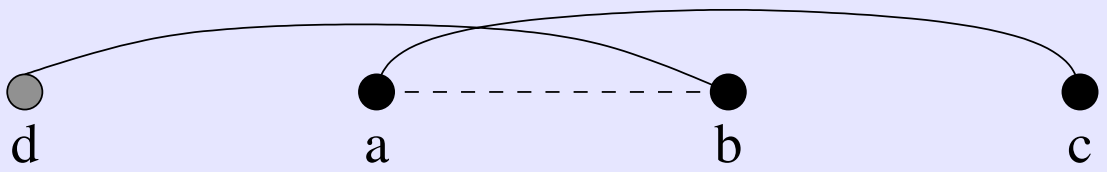
Property (BFS)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$



Property (BFS)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$



Theorem

For a graph $G = (V, E)$, an ordering σ on V is a BFS of G iff σ satisfies property (BFS).

Applications of BFS

1. Distance computations (unit length), diameter and centers,

Applications of BFS

1. Distance computations (unit length), diameter and centers,
2. BFS provides a useful layered structure of the graph

Applications of BFS

1. Distance computations (unit length), diameter and centers,
2. BFS provides a useful layered structure of the graph
3. Using BFS to search an augmenting path provides a polynomial implementation of Ford-Fulkerson maximum flow algorithm.

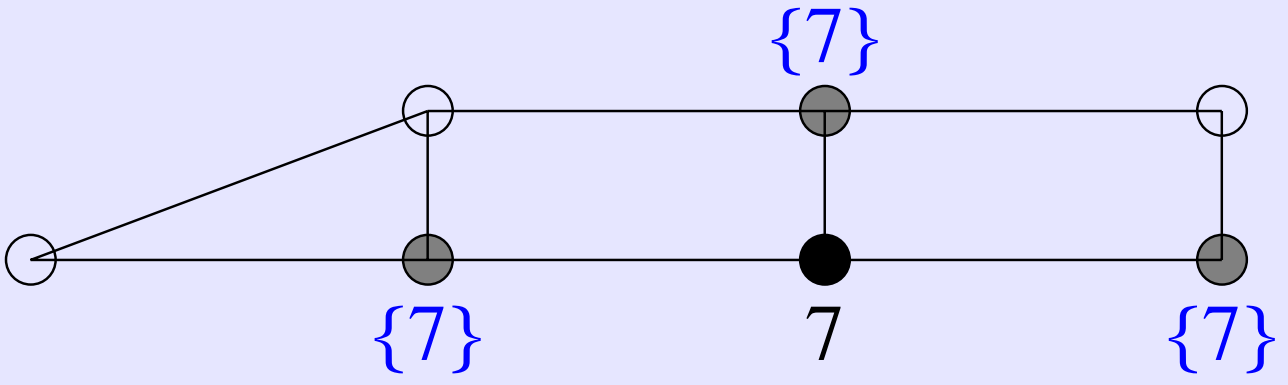
Lexicographic Breadth First Search (LBFS)

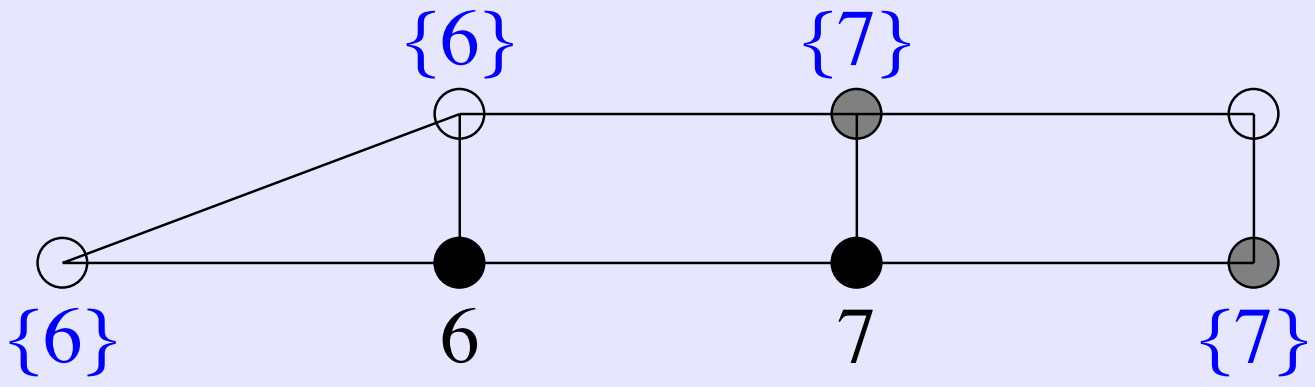
Algorithm 2: LBFS

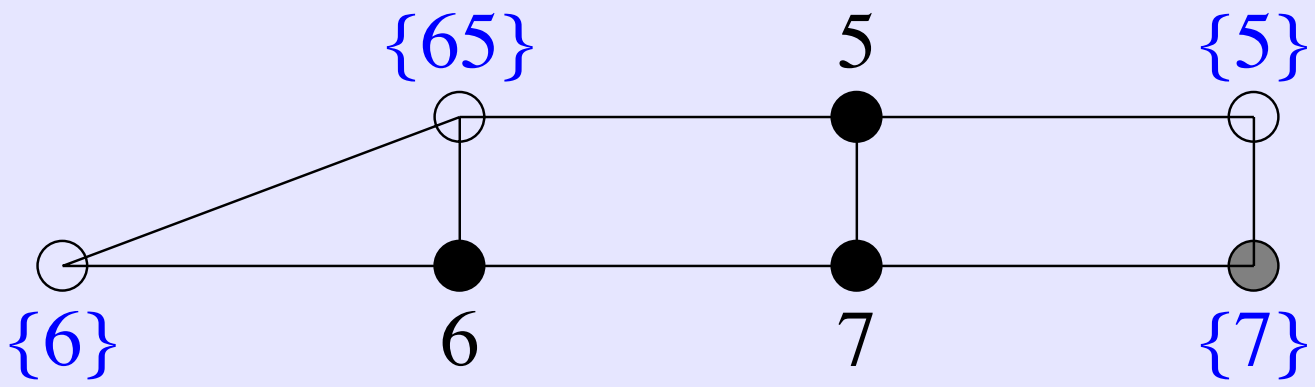
Data: a graph $G = (V, E)$ and a start vertex s

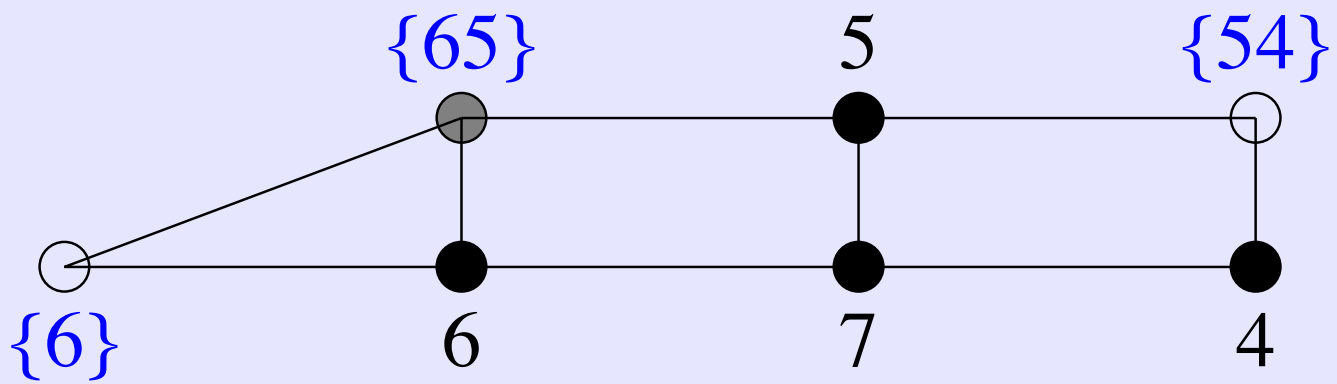
Result: an ordering σ of V

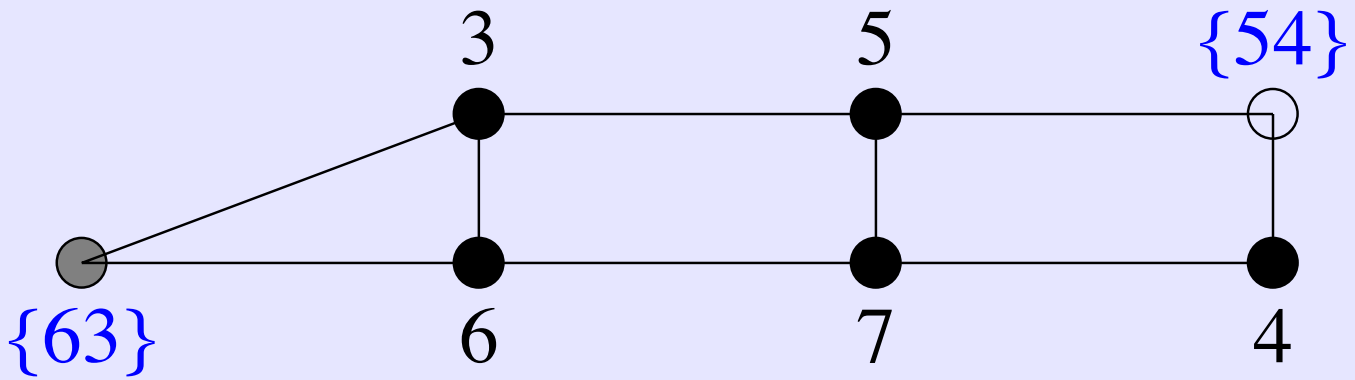
Assign the label $\emptyset$ to all vertices
$label(s) \leftarrow \{n\}$
for $i \leftarrow n$ **à** 1 **do**
    Pick an unnumbered vertex v with lexicographically largest label
    $\sigma(i) \leftarrow v$
    foreach *unnumbered vertex w adjacent to v* **do**
        $label(w) \leftarrow label(w).\{i\}$
    end
end

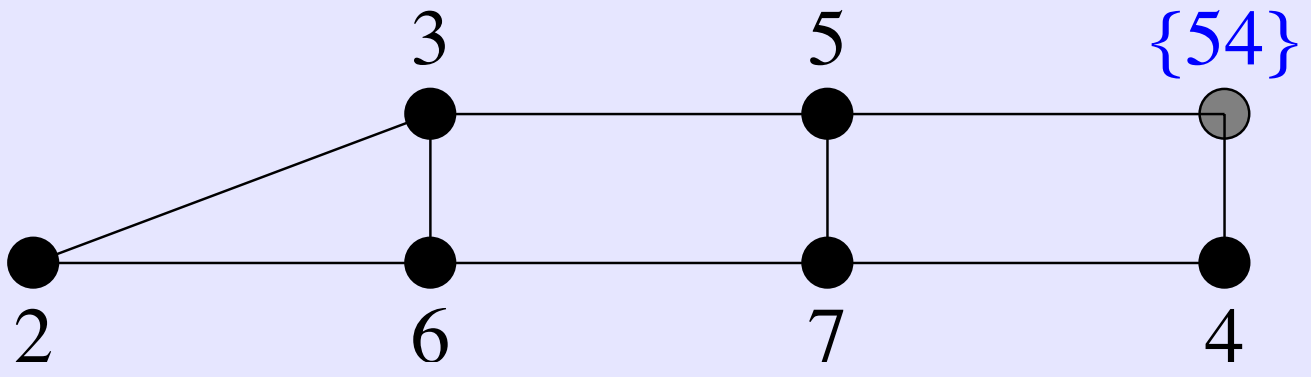

{7}
{7}
7
{7}

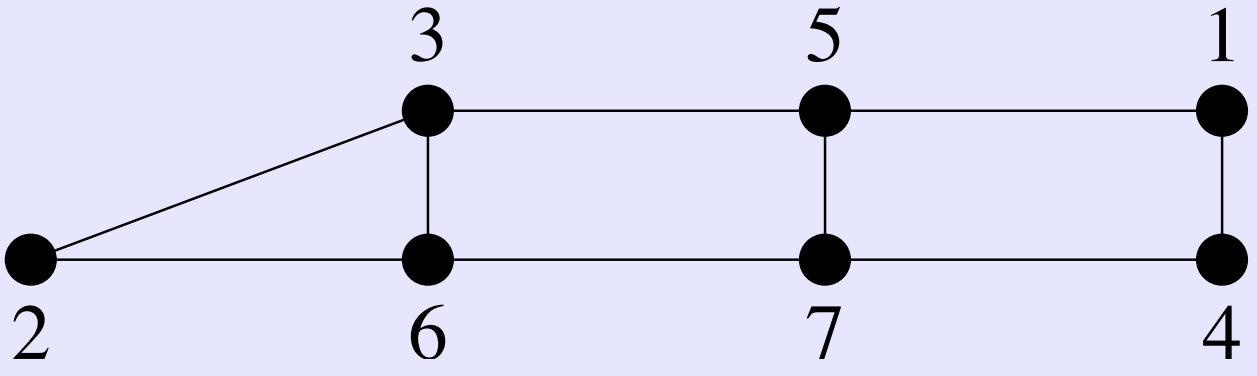

{6}
{7}
{6}
6
7
{7}


{65}
5
{5}
{6}
6
7
{7}


{65}
5
{54}
{6}
6
7
4


3
5
{54}
{63}
6
7
4


3
5
{54}
2
6
7
4

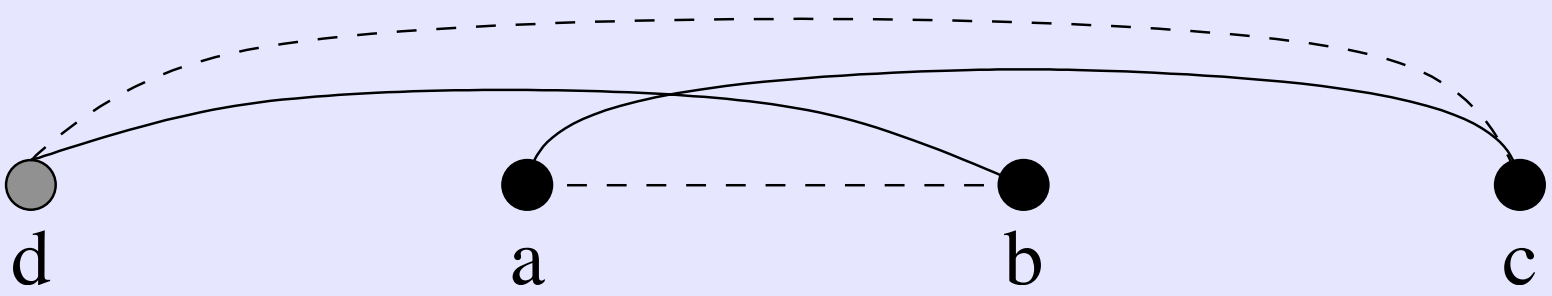

3
5
1
2
6
7
4

It is just a breadth first search with a tie break rule.

We are now considering a characterization of the order in which a LBFS explores the vertices.

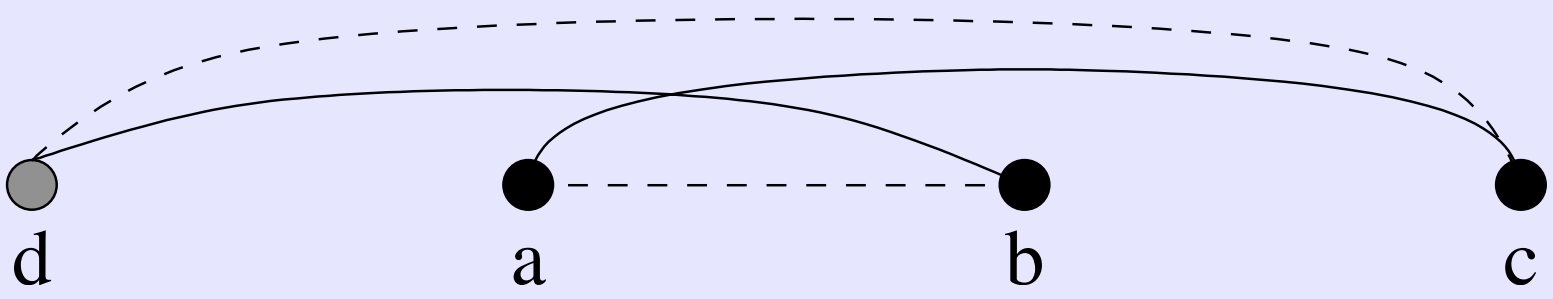
Property (LexB)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$ et $dc \notin E$.



Property (LexB)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$ et $dc \notin E$.



Theorem

For a graph $G = (V, E)$, an ordering σ on V is a LBFS of G iff σ satisfies property (LexB).

Why LBFS behaves so nicely on well-structured graphs

A nice recursive property

On every tie-break set S, LBFS operates on $G(S)$ as a LBFS.

Why LBFS behaves so nicely on well-structured graphs

A nice recursive property

On every tie-break set S, LBFS operates on $G(S)$ as a LBFS.

proof

Consider $a, b, c \in S$ such that $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$ et $dc \notin E$. But then necessarily $d \in S$.

Why LBFS behaves so nicely on well-structured graphs

A nice recursive property

On every tie-break set S, LBFS operates on $G(S)$ as a LBFS.

proof

Consider $a, b, c \in S$ such that $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $d <_\sigma a$ et $db \in E$ et $dc \notin E$. But then necessarily $d \in S$.

Remark

Analogous properties are false for other classical searches.

Applications of LBFS

1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)

Applications of LBFS

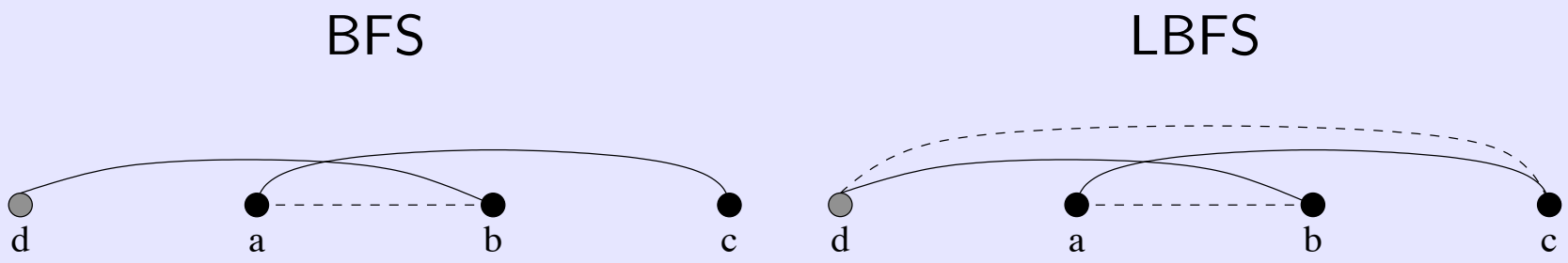
1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)
2. For many classes of graphs using LBFS ordering "backward" provides structural information on the graph.

Applications of LBFS

1. Most famous one : chordal graph recognition via simplicial elimination schemes (easy application of the 4-points condition)
2. For many classes of graphs using LBFS ordering "backward" provides structural information on the graph.
3. Last visited vertex (or clique) has some property (example last vertex is simplicial for chordal graphs)

LDFS

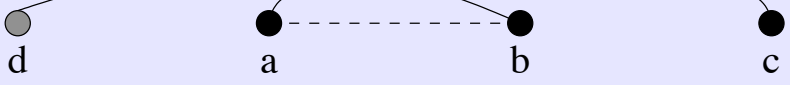
BFS vs LBFS



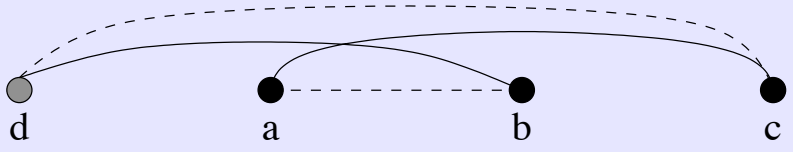
LDFS

BFS vs LBFS

BFS

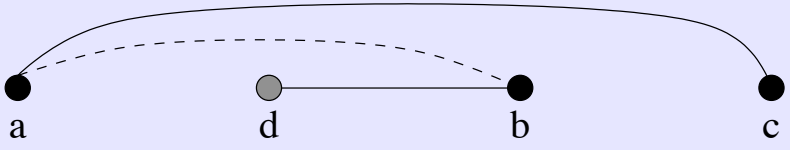


LBFS



DFS vs LDFS

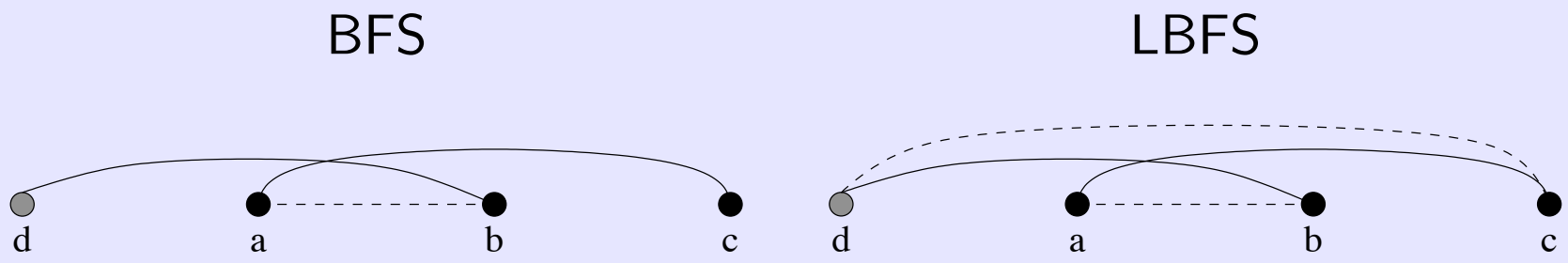
DFS



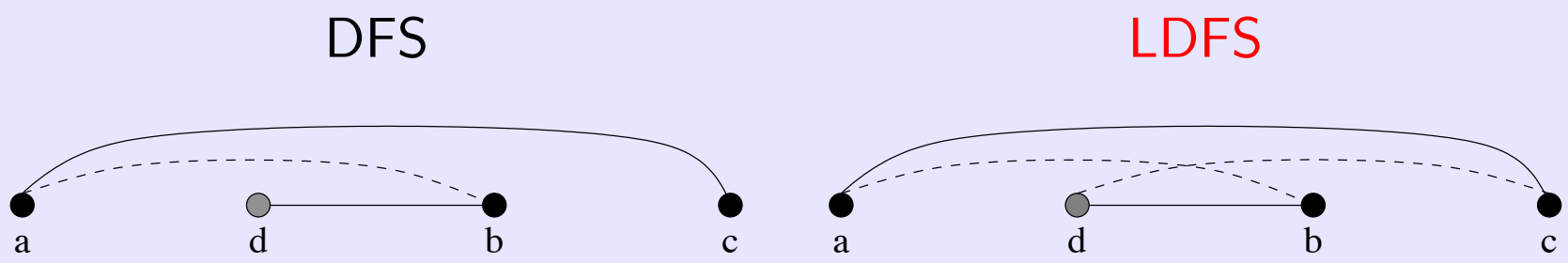
LDFS

LDFS

BFS vs LBFS

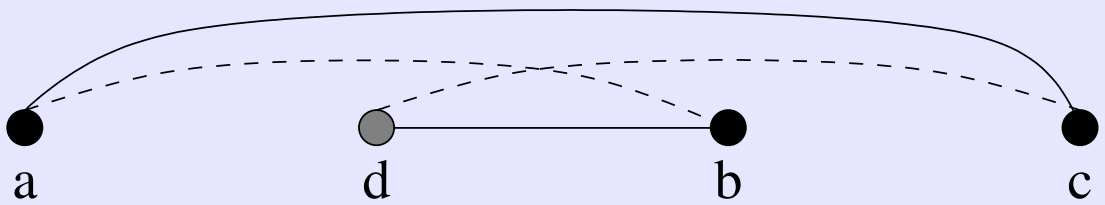


DFS vs LDFS



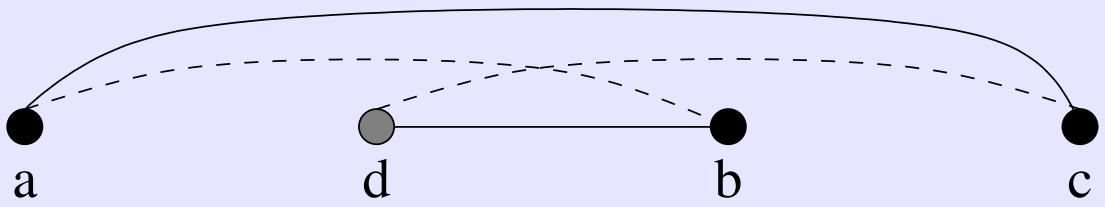
Property (LD)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $a <_\sigma d <_\sigma b$ and $db \in E$ and $dc \notin E$.



Property (LD)

For an ordering σ on V, if $a <_\sigma b <_\sigma c$ and $ac \in E$ and $ab \notin E$, then it must exist a vertex d such that $a <_\sigma d <_\sigma b$ and $db \in E$ and $dc \notin E$.



Theorem

For a graph $G = (V, E)$, an ordering σ on V is a LDFS of G iff σ satisfies property (LD).

Lexicographic Depth First Search (LDFS)

Data: a graph $G = (V, E)$ and a start vertex s

Result: an ordering σ of V

Assign the label $\emptyset$ to all vertices

$label(s) \leftarrow \{0\}$

for $i \leftarrow 1$ **à** n **do**

    Pick an unnumbered vertex v with lexicographically largest label

    $\sigma(i) \leftarrow v$

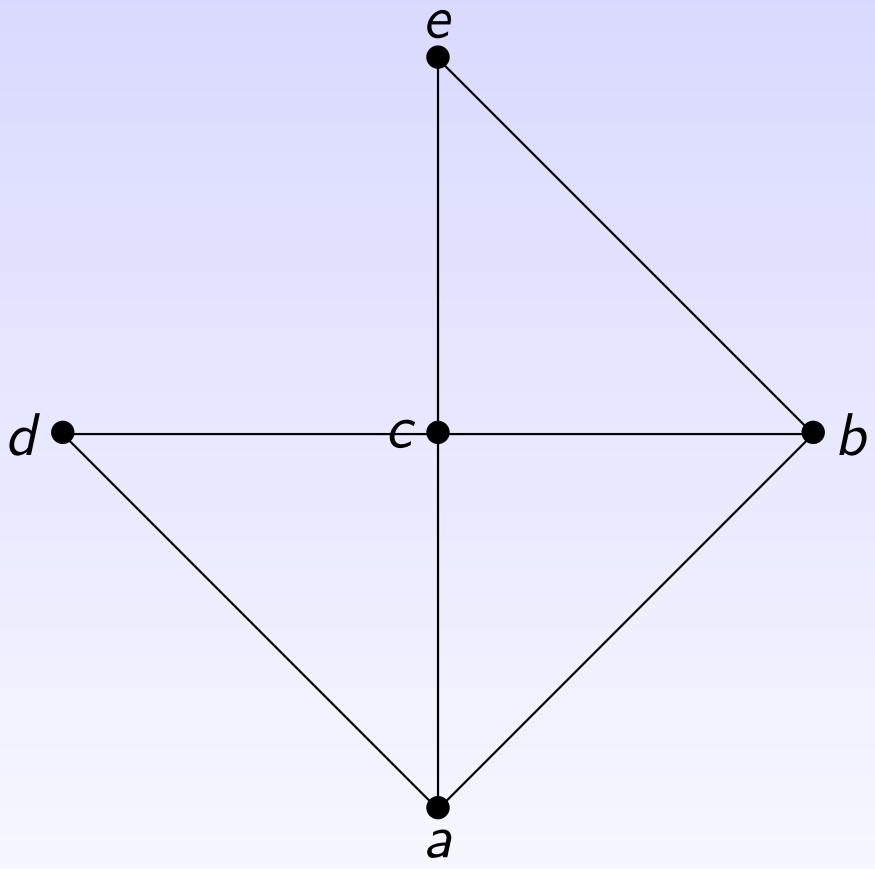
    foreach *unnumbered vertex* w *adjacent to* v **do**

        $label(w) \leftarrow \{i\}.label(w)$

    end

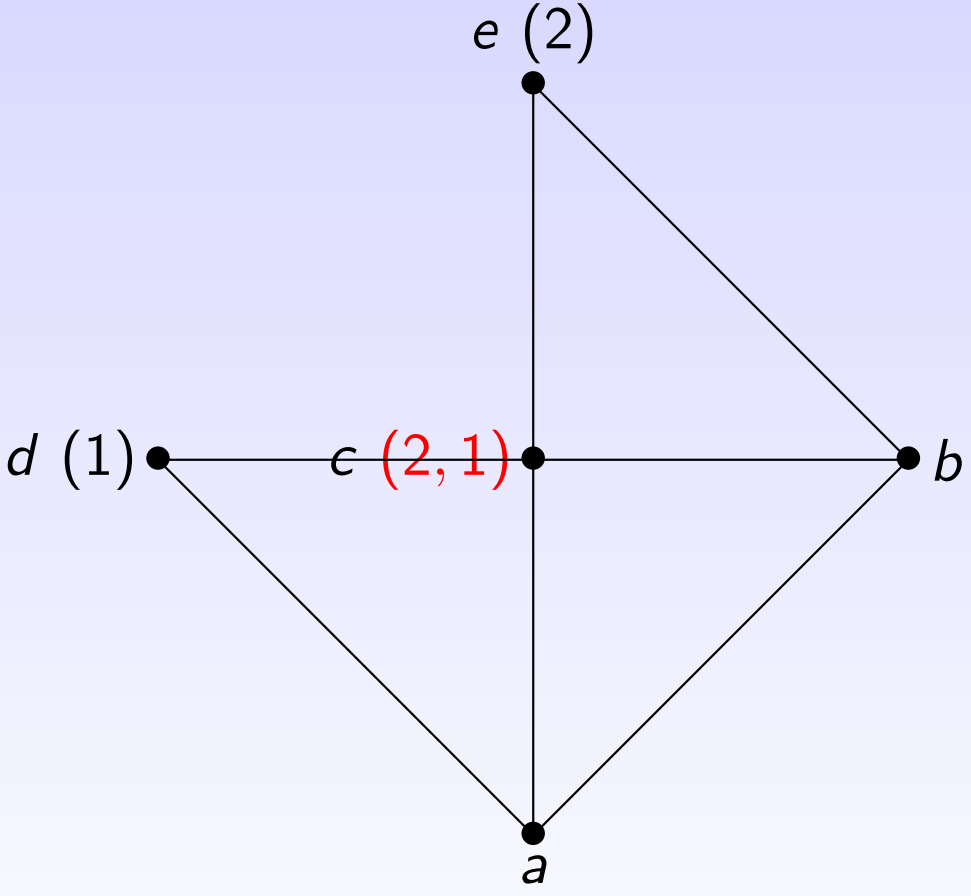
end

LDFS example



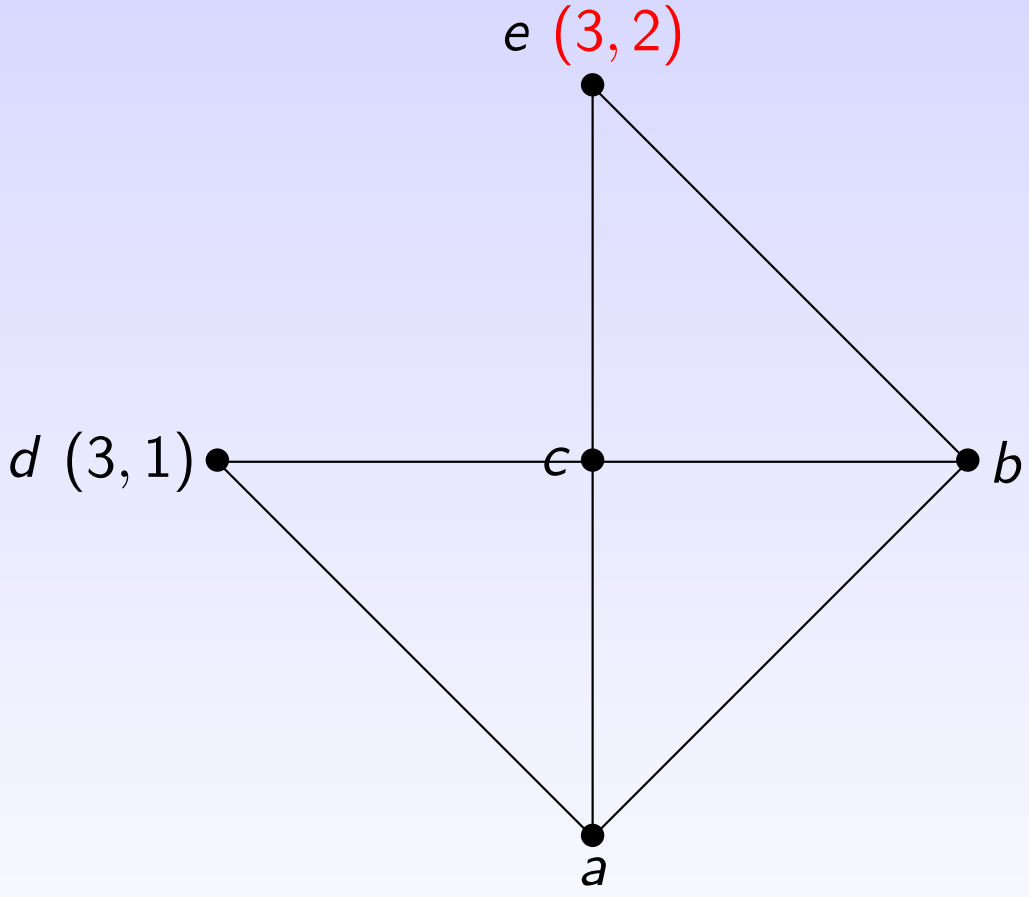
LDFS visiting a then b

LDFS example II



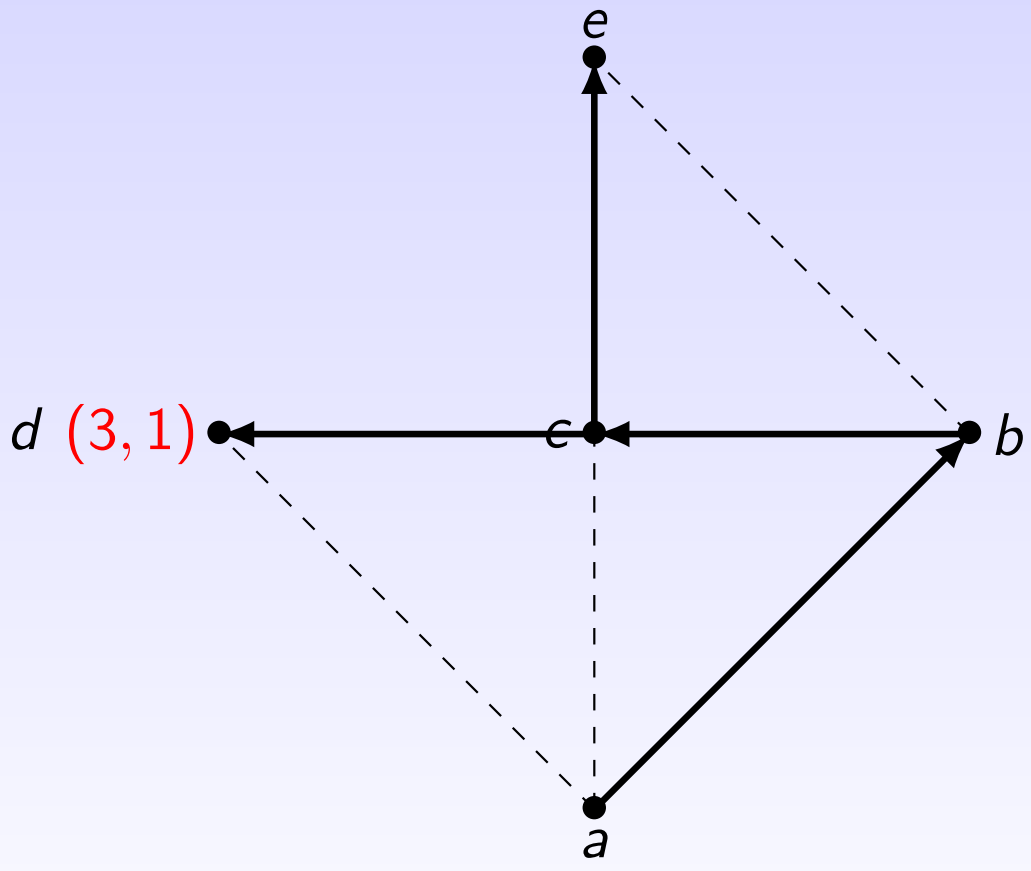
LDFS visiting a then b must visit c

LDFS example III



LDFS visiting a, b, c must visit e

LDFS example IV



LDFS visiting a, b, c, e and must finish in d

Applications of LDFS

- Hard to find application of this new tool !

Applications of LDFS

- Hard to find application of this new tool !
- Finding long paths, a very simple greedy algorithm :

Applications of LDFS

- Hard to find application of this new tool !
- Finding long paths, a very simple greedy algorithm :
- *LDFS-based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs*
  D. Corneil, B. Dalton and M. Habib **SIAM J. of Computing** 42(3) : 792-807 (2013).

Applications of LDFS

- Hard to find application of this new tool !
- Finding long paths, a very simple greedy algorithm :
- *LDFS-based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs*
  D. Corneil, B. Dalton and M. Habib **SIAM J. of Computing** 42(3) : 792-807 (2013).
- Nice graph based heuristics for community detection, J. Creusefond's work during his PhD.

So far we have

- We have considered visiting orderings of the vertices which characterize graph searches such as Generic Search, BFS, DFS, LBFS, LDFS.
- These orderings allow to prove properties on graph searches (without considering the algorithm itself)
- But also to certify (as for example for BFS and diameter computations)

Basics Definitions

Definitions :

Let G be an undirected graph :

- $exc(x) = max_{y \in G}\{distance(x, y)\}$ excentricity
- $diam(G) = max_{x \in G}\{exc(x)\}$ diameter
- $radius(G) = min_{x \in G}\{exc(x)\}$
- $x \in V$ is a center of G, if $exc(x) = radius(G)$

Basics Definitions

Definitions :

Let G be an undirected graph :

- $exc(x) = max_{y \in G}\{distance(x, y)\}$ excentricity
- $diam(G) = max_{x \in G}\{exc(x)\}$ diameter
- $radius(G) = min_{x \in G}\{exc(x)\}$
- $x \in V$ is a center of G, if $exc(x) = radius(G)$

First remarks of the definitions

distance computed in # edges
If x and y belong to different connected components $d(x, y) = \infty$.
diameter : Max Max Min
radius : Min Max Min

Trivial bounds

For any graph G :
$radius(G) \leq diam(G) \leq 2radius(G)$ and $\forall e \in G$,
$diam(G) \leq diam(G - e)$

Trivial bounds

For any graph G :
$radius(G) \leq diam(G) \leq 2radius(G)$ and $\forall e \in G$,
$diam(G) \leq diam(G - e)$

These bounds are tight

Trivial bounds

For any graph G :
$radius(G) \leq diam(G) \leq 2radius(G)$ and $\forall e \in G$,
$diam(G) \leq diam(G - e)$

These bounds are tight

- If G is a path of length 2K, then $diam(G) = 2k = 2radius(G)$, and G admits a unique center, i.e. the middle of the path.

Trivial bounds

For any graph G :
$radius(G) \leq diam(G) \leq 2radius(G)$ and $\forall e \in G$,
$diam(G) \leq diam(G - e)$

These bounds are tight

- If G is a path of length 2K, then $diam(G) = 2k = 2radius(G)$, and G admits a unique center, i.e. the middle of the path.
- If $radius(G) = diam(G)$, then $Center(G) = V$. All vertices are centers (as for example in a cycle).
  It will remain as the typical bad graph structure for diameter computations.

If $2.radius(G) = diam(G)$, then *roughly* G has a tree shape (at least it works for trees).
But there is no nice characterization of this class of graphs.

Diameter

Applications

1. A graph parameter which measures the quality of services of a network, in terms of worst cases, when all have a unitary cost. Find critical edges e s.t. $diam(G - e) > diam(G)$
2. Many distributed algorithms can be analyzed with this parameter (when a flooding technique is used to spread information over the network or to construct routing tables).
3. Verify the small world hypothesis in some large social networks, using J. Kleinberg's definition of small world graphs.
4. Compute the diameter of the Internet graph, or some Web graphs, i.e. massive data.

1. Examples of diameter searches based on the algorithms presented in this lecture : `https://files.inria.fr/gang/road/`
2. OpenStreetMap (OSM) : 80 millions of nodes, average degree 3
3. Roadmaps graphs a special domain of research interest Quasi-planar graph (bridges on the roads)

Frequently Asked Questions (FAQ)

Usual questions on diameter, centers and radius :

- What is the best Program (resp. algorithm) available ?
- What is the complexity of diameter, center and radius computations ?
- How to compute or approximate the diameter of huge graphs ?
- Find a center (or all centers) in a network, (in order to install serveurs).

- Our aim is to design an algorithm or heuristic to compute the diameter of very large graphs
- Any algorithm that computes all distances between all pairs of vertices, complexity $O(n^3)$ or $O(nm)$. As for example with $|V|$ successive Breadth First Searches in $O(n(n+m))$.
- Best known complexity for an exact algorithm is $O(\frac{n^3}{log^3 n})$, in fact computing all shortest paths.
- But also with at most $O(Diam(G))$ matrix multiplications.

New game : computing diameter using fewest BFS possible

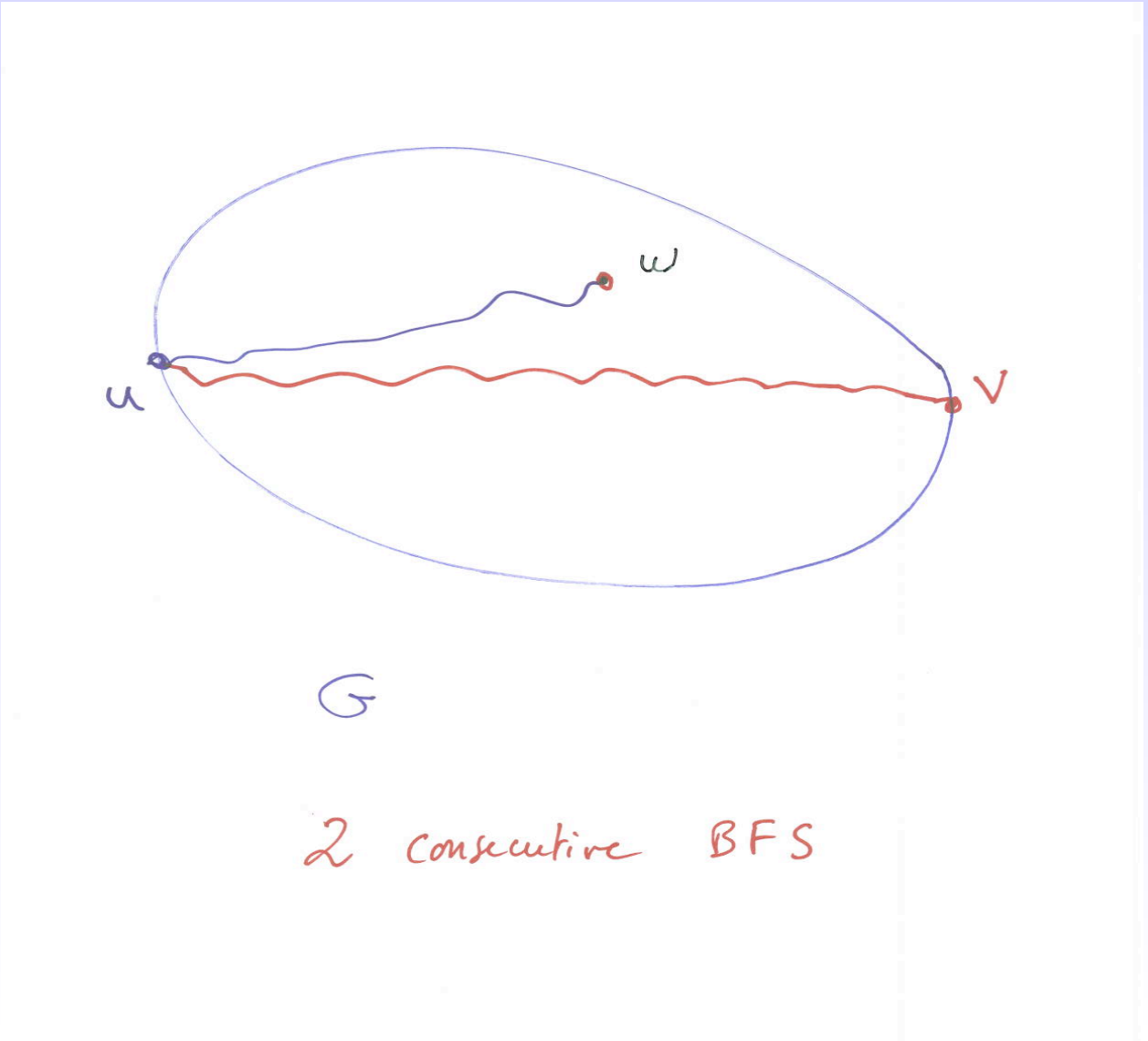
1. Let us consider the procedure called : 2 consecutive BFS [1]

Data: A graph $G = (V, E)$
Result: u, v two vertices
Choose a vertex $w \in V$
$u \leftarrow BFS(w)$
$v \leftarrow BFS(u)$

Therefore it is a linear procedure

1. Proposed the first time by Handler 1973

Intuition behind the procedure



- Handler's classical result 73
  If G is a tree, $diam(G) = d(u, v)$
  Easy using Jordan's theorem.

First theorem

Camille Jordan 1869 :

A tree admits one or two centers depending on the parity of its diameter and furthermore all chains of maximum length starting at any vertex contain this (resp. these) centers.
And $radius(G) = \lceil \frac{diam(G)}{2} \rceil$
Camille Jordan, *Sur les assemblages de lignes*, Journal für reine und angewandte Mathematik **70** (1869), 185–190.

Unfortunately it is not an algorithm !

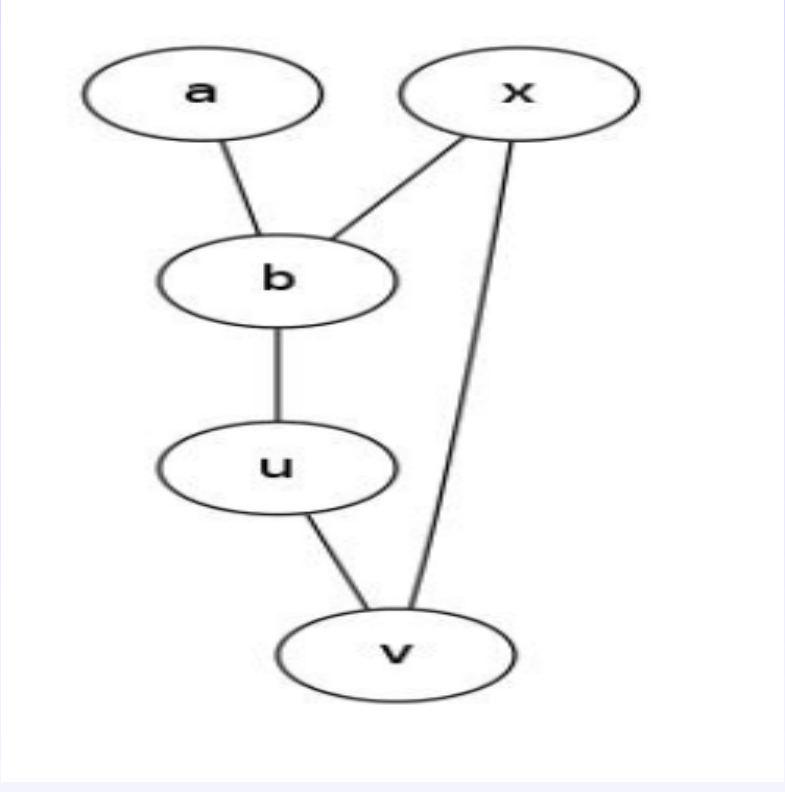


FIGURE – $u \leftarrow BFS(x) = \{x, b, v, a, u\}$ and
$a \leftarrow BFS(u) = \{u, b, v, x, a\}$. But $diam(G) = 3$ with $[a, b, u, v]$

Experimental results : M.H., M.Latapy, C. Magnien 2008

Randomized BFS procedure

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

Result: u, v two vertices

Repeat α times :

Randomly Choose a vertex $w \in V$

$u \leftarrow BFS(w)$

$v \leftarrow BFS(u)$

Select the vertices u_0, v_0 s.t. $distance(u_0, v_0)$ is maximal.

---

1. This procedure gives a vertex u_0 such that :
   $exc(u_0) \leq diam(G)$ i.e. a lower bound of the diameter.

1. This procedure gives a vertex u_0 such that : $exc(u_0) \leq diam(G)$ i.e. a lower bound of the diameter.
2. Use a spanning tree as a subgraph on the same vertex set to obtain an upper bound by computing its exact diameter in linear time (using the trivial bound $diam(G) \leq diam(G - e)$).

1. This procedure gives a vertex u_0 such that : $exc(u_0) \leq diam(G)$ i.e. a lower bound of the diameter.
2. Use a spanning tree as a subgraph on the same vertex set to obtain an upper bound by computing its exact diameter in linear time (using the trivial bound $diam(G) \leq diam(G - e)$).
3. Spanning trees given by the BFS.

- Since α is a constant (≤ 1000), this method is still in linear time and works extremely well on huge graphs (Web graphs, Internet ...)

- Since α is a constant (≤ 1000), this method is still in linear time and works extremely well on huge graphs (Web graphs, Internet ...)
- How can we explain the success of such a method ?

- Since α is a constant (≤ 1000), this method is still in linear time and works extremely well on huge graphs (Web graphs, Internet ...)
- How can we explain the success of such a method ?
- Due to the many counterexamples for the 2 consecutive BFS procedure. An explanation is necessary !

Chordal graphs

1. A graph is chordal if it has no chordless cycle of length ≥ 4 .

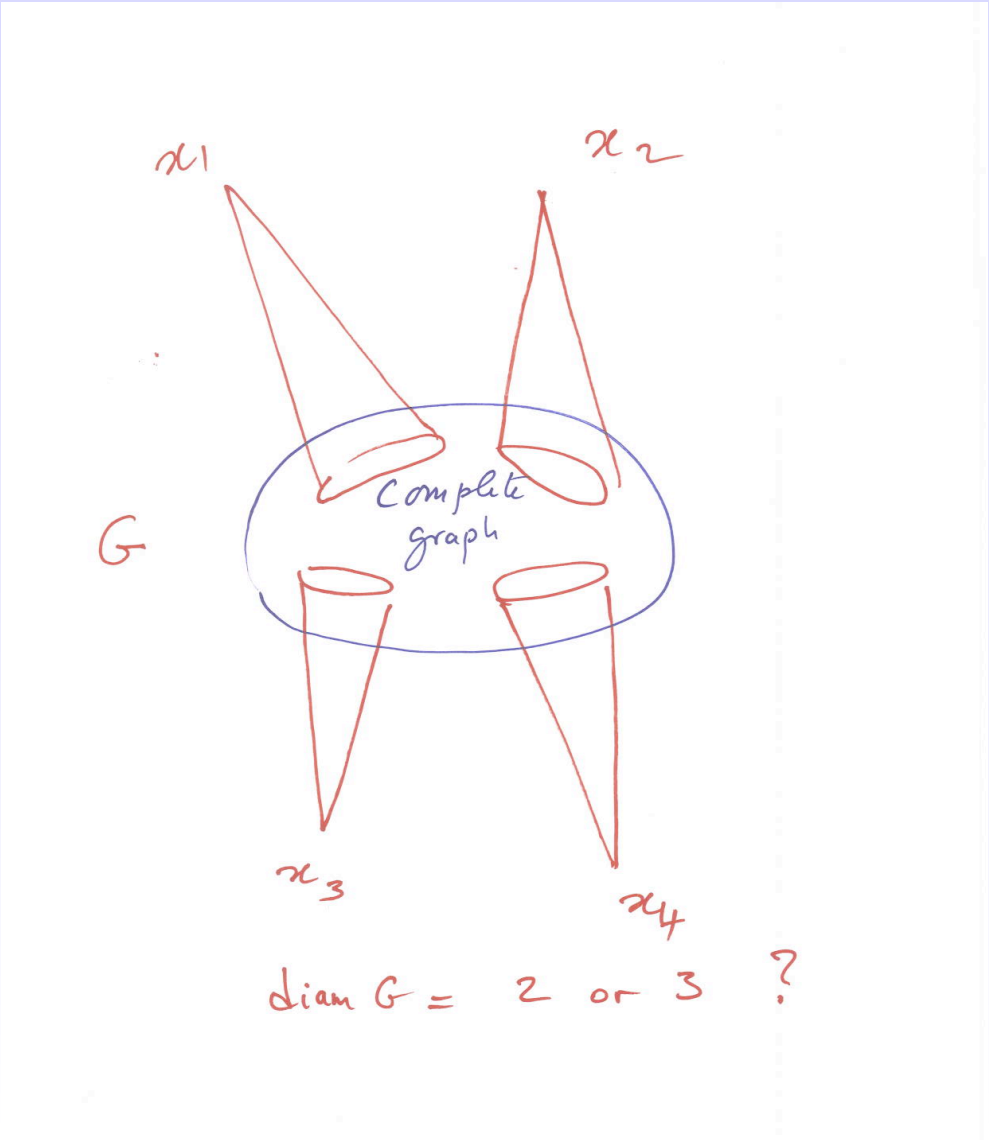
Chordal graphs

1. A graph is chordal if it has no chordless cycle of length ≥ 4 .
2. If G is a chordal graph, Corneil, Dragan, H., Paul 2001, using a variant called 2 consecutive LexBFS
   $d(u, v) \leq diam(G) \leq d(u, v) + 1$

Chordal graphs

1. A graph is chordal if it has no chordless cycle of length ≥ 4 .
2. If G is a chordal graph, Corneil, Dragan, H., Paul 2001, using a variant called 2 consecutive LexBFS
   $d(u, v) \leq diam(G) \leq d(u, v) + 1$
3. Generalized by Corneil, Dragan, Kohler 2003 using 2 consecutive BFS :
   $d(u, v) \leq diam(G) \leq d(u, v) + 1$

Chordal graphs and split graphs



Disjoint sets problem

Disjoint sets problem

A finite set X, $\mathcal{F}$ a collection $\{S_1, \ldots, S_k\}$ of subsets of X.
$\exists i, j \in [1, k]$ such that $S_i \cap S_j = \emptyset$?

Disjoint sets problem

Disjoint sets problem

A finite set X, $\mathcal{F}$ a collection $\{S_1, \ldots, S_k\}$ of subsets of X.
$\exists i, j \in [1, k]$ such that $S_i \cap S_j = \emptyset$?

Linearity

Can this problem be solved in linear time ?
Size of the problem : $|X| + k + \sum_{i=1}^{i=k} |S_i|$
size of the incidence bipartite graph

SETH : Strong Exponential Time Hypothesis

Computing diameter is now a hot subject due to lower bounds techniques

SETH

There is no algorithm for solving the k-SAT problem with n variables in $O((2)^{n-\epsilon})$ where ϵ does not depend on k.

Let us consider an instance I of k-SAT with 2n boolean variables $x_1, \ldots, x_{2n}$, and a set of m clauses $\mathcal{C} = \{C_1, \ldots C_m\}$, we build an instance of Disjoint-set problem as follows :

- The gound set X is the set of clauses + 2 extras vertices a, b.

Let us consider an instance I of k-SAT with 2n boolean variables $x_1, \ldots, x_{2n}$, and a set of m clauses $\mathcal{C} = \{C_1, \ldots C_m\}$, we build an instance of Disjoint-set problem as follows :

- The gound set X is the set of clauses + 2 extras vertices a, b.
- We consider now A, B the sets of all truth assignments of $x_1, \ldots, x_n$, and $x_{n+1}, \ldots x_{2n}$, respectively.

Let us consider an instance I of k-SAT with 2n boolean variables $x_1, \ldots, x_{2n}$, and a set of m clauses $\mathcal{C} = \{C_1, \ldots C_m\}$, we build an instance of Disjoint-set problem as follows :

- The gound set X is the set of clauses + 2 extras vertices a, b.
- We consider now A, B the sets of all truth assignments of $x_1, \ldots, x_n$, and $x_{n+1}, \ldots x_{2n}$, respectively.
- For each truth t assigment in A (resp. in B) we define $S_t = \{C \in \mathcal{C}$ such that t does not satisfy $C\} \cup \{a\}$ (resp. $\cup\{b\}$).

- The sets $S's$ defined with A (resp. B) always intersect because of a (resp. b).

- The sets $S's$ defined with A (resp. B) always intersect because of a (resp. b).
- If there exists S_u, S_v that do not intersect. Necessarily u is a truth assignment in A and v in B (or the converse, but they cannot be on the same set of variables because of the dummy vertices a, b).
  This means that for each clause C_i of I, if $C_i \notin S_u$, then the truth v assignment satisfies C_i.
  Similarly if $C_i \notin S_v$, then the truth u assignment satisfies C_i.
  But $S_u \cap S_v = \emptyset$ means that for every clause C_i either :
  $C_i \notin S_u$ or $C_i \notin S_v$.

- The sets $S's$ defined with A (resp. B) always intersect because of a (resp. b).
- If there exists S_u, S_v that do not intersect. Necessarily u is a truth assignment in A and v in B (or the converse, but they cannot be on the same set of variables because of the dummy vertices a, b).
  This means that for each clause C_i of I, if $C_i \notin S_u$, then the truth v assignment satisfies C_i.
  Similarly if $C_i \notin S_v$, then the truth u assignment satisfies C_i.
  But $S_u \cap S_v = \emptyset$ means that for every clause C_i either :
  $C_i \notin S_u$ or $C_i \notin S_v$.
- Therefore :
  I is satisfiable iff there exist 2 disjoint sets S_u, S_v.

Complexity issues

- Size of the k-SAT instance is bounded by :
  $K = 2n + m + km$

Complexity issues

- Size of the k-SAT instance is bounded by :
  $K = 2n + m + km$
- Size of the Disjoint set instance :
  $N = 2^{n+1} + m + 2$ vertices
  and at most $M = m2^{n+1}$ edges.

Complexity issues

- Size of the k-SAT instance is bounded by :
  $K = 2n + m + km$
- Size of the Disjoint set instance :
  $N = 2^{n+1} + m + 2$ vertices
  and at most $M = m2^{n+1}$ edges.
- To compute this instance we need to evaluate the m, k-clauses for each half-truth assigment.
  Can be done in $O(K)$, so in the whole : $O(2^{n+1}K)$.

Complexity issues

- Size of the k-SAT instance is bounded by :
  $K = 2n + m + km$
- Size of the Disjoint set instance :
  $N = 2^{n+1} + m + 2$ vertices
  and at most $M = m2^{n+1}$ edges.
- To compute this instance we need to evaluate the m, k-clauses for each half-truth assigment.
  Can be done in $O(K)$, so in the whole : $O(2^{n+1}K)$.
- If there exists an algorithm for the Disjoint set problem in less than $O(NM^{1-\epsilon})$
  it would imply an algorithm for k-SAT in less than $O(2^{2n})$
  contradicting the SETH.

Consequences

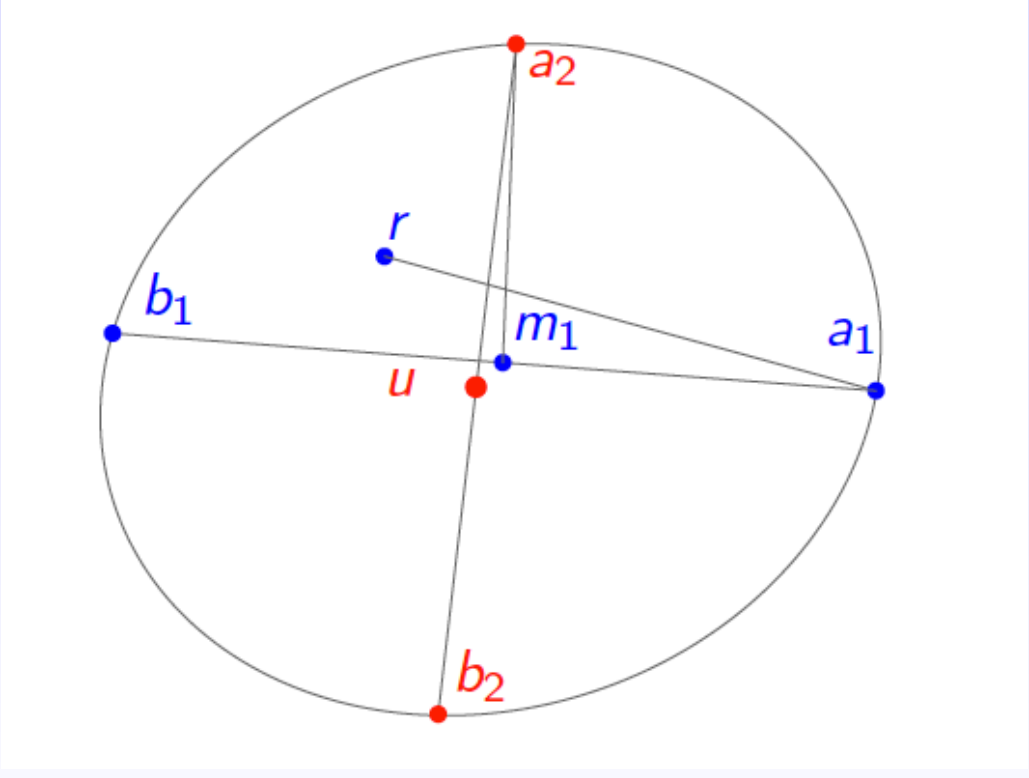
It could be difficult to design a linear time algorithm for :

1. Disjoint set problem
2. Diameter computations for chordal graphs and split graphs
3. And many other related problems . . . such as betweenness centrality
4. but not all $O(mn)$ problems as for example transitive closure, existence of a triangle . . .

Research Problem

- Since sparse graphs are not available for the above reduction.
- Can we compute in linear time the diameter of planar graphs ?

The 4-sweep method : Crescenzi, Grossi, MH, Lanzi, Marino 2011



$Diam = max\{ecc(a_1), ecc(a_2)\}$ and $Rad = min\{ecc(r), ecc(m_1)\}$

Intuition behind the 4-sweep heuristics

- Chepoi and Dragan has proved that for chordal graphs that a center is at distance at most one of the middle vertex (m_1 in the picture). Knowledge on graph classes can be useful

Intuition behind the 4-sweep heuristics

- Chepoi and Dragan has proved that for chordal graphs that a center is at distance at most one of the middle vertex (m_1 in the picture). Knowledge on graph classes can be useful
- Roughly, we have the same results with 4-sweep than with 1000 2-sweep.

It is still not al algorithm ! !

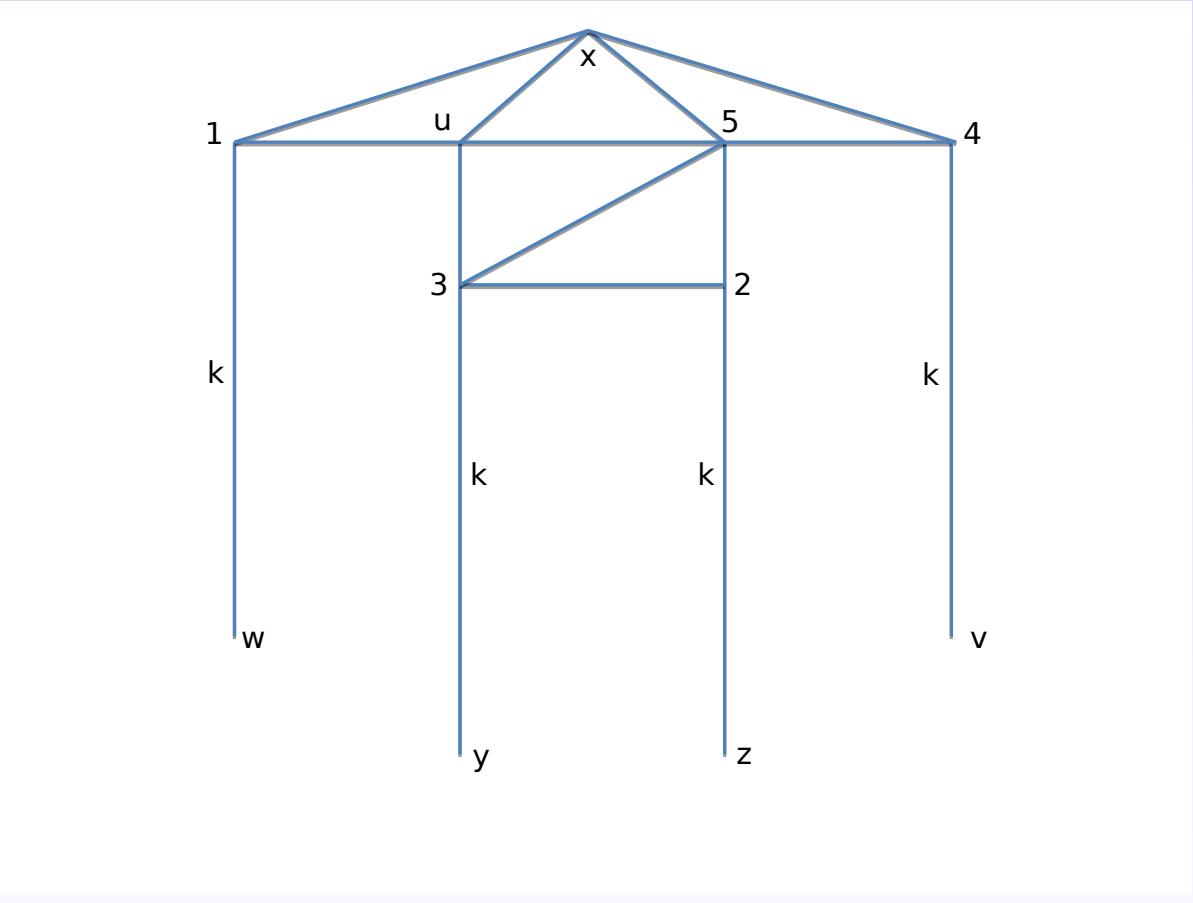


FIGURE – First [y,v], then [v,w] and [x,y], [y,v] max=2k+2, so [z,w] of length 2k+3 is never reached.

Stanford Large Network Dataset Collection

http ://snap.stanford.edu/data/

- A very practical database for having large graphs to play with.

Stanford Large Network Dataset Collection

http ://snap.stanford.edu/data/

- A very practical database for having large graphs to play with.
- Graphs are described that way : number of vertices, number of edges (arcs), diameter.

Graph	diam SNAP	diam 4-Sweep
soc-Epinions1	14	15
soc-pokec-relationships	11	14
soc-Slashdot0811	10	12
soc-Slashdot0902	11	13
com-lj.ungraph	17	21
com-youtube.ungraph	20	24
com-DBLP	21	23
com-amazon	44	47
email-Enron	11	13
wikiTalk	9	11
cit-HepPh	12	14
cit-HepTh	13	15
CA-CondMat	14	15
CA-HepTh	17	18
web-Google	21	24

Graph	diam SNAP	diam 4-Sweep
amazon0302	32	38
amazon0312	18	20
amazon0505	20	22
amazon0601	21	25
p2p-Gnutella04	9	10
p2p-Gnutella24	10	11
p2p-Gnutella25	10	11
p2p-Gnutella30	10	11
roadNet-CA	849	865
roadNet-TX	1054	1064
Gowalla-edges	14	16
BrightKite-edges	16	18

- How can we beat the value of Stanford database ?

- How can we beat the value of Stanford database ?
- In fact some * explains in a little footnote that the SNAP value is heuristically obtained by 1000 random BFS

- How can we beat the value of Stanford database ?
- In fact some * explains in a little footnote that the SNAP value is heuristically obtained by 1000 random BFS
- I like the idea that 4 searches totally dependant are better that 1000 independant searches

- How can we beat the value of Stanford database ?
- In fact some * explains in a little footnote that the SNAP value is heuristically obtained by 1000 random BFS
- I like the idea that 4 searches totally dependant are better that 1000 independant searches
- See the example of a long path.

- How can we beat the value of Stanford database ?
- In fact some * explains in a little footnote that the SNAP value is heuristically obtained by 1000 random BFS
- I like the idea that 4 searches totally dependant are better that 1000 independant searches
- See the example of a long path.
- The last vertex of a BFS is not at all a random vertex (NP-complete to decide : Charbit, MH, Mamcarz 2014)

How can we certify our results ?

- By certifying the longest path $[x, y]$ (as hard as computing a BFS ?)

How can we certify our results ?

- By certifying the longest path $[x, y]$ (as hard as computing a BFS ?)
- Using another BFS programmed by others starting at x.

How can we certify our results ?

- By certifying the longest path $[x, y]$ (as hard as computing a BFS ?)
- Using another BFS programmed by others starting at x.
- Certifying that the computed BFS ordering is a legal BFS ordering, using the 4-point condition. Which can be checked in linear time for BFS and DFS.

Graph Name	$\frac{Vertices}{Edges}$	Real Diameter	Diam. FourSweep
CA-HepTh	0.190	18	18
CA-GrQc	0.181	17	17
CA-CondMat	0.124	15	15
CA-AstroPh	0.047	14	14
roadNet-CA	0.355	865	865
roadNet-PA	0.353	794	780
roadNet-TX	0.359	1064	1064
email-Enron	0.1	13	13
email-EuAll	0.631	14	14
com-amazon	0.361	47	47
Amazon0302	0.212	38	38
Amazon0312	0.125	20	20
Amazon0505	0.122	22	22
Amazon0601	0.119	25	25
Gowalla_edges	0.207	25	16
Brightkite_edges	0.272	18	18
soc-Epinions1	0.149	15	15

FIGURE – 4-Sweep Results

An old question to Vlady :

What can you say about the Proba(4-sweep is correct) ?

Further results

- IFUB an algorithm to compute the excentricity in a bottom up fashion starting from the leaves of a BFS rooted in m_1 with a stoping condition.
  Complexity is $O(nm)$ in the worst case, but often linear in practice.

Comments

- Boldi and his group had to parallelize our algorithm and a BFS on the giant connected component of Facebook would take several hours. But only 17 BFS's were needed.
  Diametre Facebook = 41 !, Average distance 4.74, Backstrom, Boldi, Rosa, Uganden, Vigna 2011

Comments

- Boldi and his group had to parallelize our algorithm and a BFS on the giant connected component of Facebook would take several hours. But only 17 BFS's were needed.
  Diametre Facebook = 41 !, Average distance 4.74, Backstrom, Boldi, Rosa, Uganden, Vigna 2011
- The 4-sweep method alway gives a lower bound of the diameter not too far from the optimal,
  the hard part is to obtain an upper bound with iFUB

Comments

- Boldi and his group had to parallelize our algorithm and a BFS on the giant connected component of Facebook would take several hours. But only 17 BFS's were needed.
  Diametre Facebook = 41 !, Average distance 4.74, Backstrom, Boldi, Rosa, Uganden, Vigna 2011
- The 4-sweep method alway gives a lower bound of the diameter not too far from the optimal,
  the hard part is to obtain an upper bound with iFUB
- The worst examples are roadmap graphs with big treewidth and big grids.

A method symmetric for computing radius and diameter

M. Borassi, P. Crescenzi, R. Grossi, M.H., W. Kosters, A. Marino and F. Takes, 2014

- A mixture with our approach and that of W. Kosters and F. Takes in which a lower bound of the eccentricity of every vertex is maintained at each BFS.

A method symmetric for computing radius and diameter

M. Borassi, P. Crescenzi, R. Grossi, M.H., W. Kosters, A. Marino and F. Takes, 2014

- A mixture with our approach and that of W. Kosters and F. Takes in which a lower bound of the eccentricity of every vertex is maintained at each BFS.
- It generalizes the 4-sweep to k-sweep.

A method symmetric for computing radius and diameter

M. Borassi, P. Crescenzi, R. Grossi, M.H., W. Kosters, A. Marino and F. Takes, 2014

- A mixture with our approach and that of W. Kosters and F. Takes in which a lower bound of the eccentricity of every vertex is maintained at each BFS.
- It generalizes the 4-sweep to k-sweep.
- we generalize to maintain k values in each vertex.

Sketch of the algorithm

- Given a random vertex v_1 and setting $i = 1$, repeat the following :
  1. Perform a BFS from v_i and choose the vertex v_{i+1} as the vertex x maximizing $\sum_{j=1}^{i} d(v_j, x)$.
  2. Compute the eccentricity of v_{j+2}, the vertex minimizing $\sum_{j=1}^{i+1} d(w, v_j)$.
  3. Increment i by 2.
  4. Maintain bounds on each vertex for an halting condition
- The maximum eccentricity found, i.e. $\max_{j=1,\ldots,i} exc(v_j)$, is a lower bound for the diameter.
- The minimum eccentricity found, i.e. $\min\{\min_{j=1,\ldots,i} exc(v_j)\}$, is an upper bound for the radius.

Recent results 2018

Feodor Dragan, MH, Laurent Viennot 2018
Yesterday on archiv. We give an explanation of the efficiency of these algorithms
bounding the complexity in terms of LastBFS vertices.

Real Applications

With this method we were able to disprove conjectures inspired from S. Milgram about the 6 degrees of separation

1. Kevin Bacon games on the actors graph

Real Applications

With this method we were able to disprove conjectures inspired from S. Milgram about the 6 degrees of separation

1. Kevin Bacon games on the actors graph
2. Diameter of Wikipedia (the Wiki Game)

Kevin Bacon

His name was used for a popular TV game in US, The Six Degrees of Kevin Bacon, in which the goal is to connect an actor to Kevin Bacon in less than 6 edges.

Actors graph 2014

- The 2014 graph has 1.797.446 in the biggest connected component, a few more if we consider the whole graph. The number of undirected edges in the biggest connected component is 72.880.156.

Actors graph 2014

- The 2014 graph has 1.797.446 in the biggest connected component, a few more if we consider the whole graph. The number of undirected edges in the biggest connected component is 72.880.156.
- An actor with Bacon number 8 is Shemise Evans, and the path can be found at http ://oracleofbacon.org/ by writing Shemise Evans in the box. Even if their graph does not coincide exactly with our graph, this is a shortest path in both of them :

Shemise Evans → Casual Friday (2008) → Deniz Buga
Deniz Buga → Walking While Sleeping (2009)→ Onur Karaoglu
Onur Karaoglu→ Kardesler (2004)→ Fatih Genckal
Fatih Genckal → Hasat (2012) → Mehmet Ünal
Mehmet Ünal→ Kayip özgürlük (2011)→ Aydin Orak
Aydin Orak → The Blue Man (2014)→Alex Dawe
Alex Dawe→ Taken 2 (2012)→ Rade Serbedzija
Rade Serbedzija→ X-Men : First Class (2011) → Kevin Bacon

Twitter graph 2011

Directed Graph of 500 millions of nodes
2,5 billiard of edges
150 diameter of the giant strongly connected component.

Theoretical aspects

- D. Corneil, F. Dragan, M. Habib, C. Paul, *Diameter determination on restricted families of graphs*, **Discrete Applied Mathematic**, Vol 113(2-3) : 143-166 (2001)
- V. Chepoi, F. Dragan, B. Estellon, M. Habib, Y. Vaxes, *Diameters, centers, and approximating trees of delta-hyperbolic geodesic spaces and graphs*, **ACM Symposium on Computational Geometry** 2008 : 59-68.
- V. Chepoi, F. Dragan, B. Estellon, M. Habib, Y. Vaxes, *Notes on diameters, centers, and approximating trees of δ-hyperbolic geodesic spaces and graphs*, TGCT08 Paris, **Electronic Notes in Discrete Mathematics** 31(2008)231-234.
- V. Chepoi, F. Dragan, B. Estrellon, M. Habib, Y. Vaxes et Y. Xiang *Additive Spanners and Distance and Routing Labeling Schemes for Hyperbolic Graphs*, **Algorithmica** 62-(3-4) (2012) 713-732.

Algorithmic and experimental aspects

- C. Magnien, M. Latapy, M. Habib, *Fast computation of empirically tight bounds for the diameter of massive graphs*, **Journal of Experimental Algorithmics**, 13 (2008).
- P. Crescenzi, R. Grossi, M. Habib, L. Lanzi and A. Marino, *On Computing the Diameter of Real-World Undirected graphs*, **Theor. Comput. Sci.** 514 : 84-95 (2013).
- M. Borassi, P. Crescenzi, R. Grossi, M. Habib, W. Kosters, A. Marino and F. Takes, *Fast diameter and radius BFS-based computation in (weakly connected) real-world graphs : With an application to the six degrees of separation games*, **Theor. Comput. Sci.** 586 : 59-80 (2015)
- F. Dragan, M. Habib, L. Viennot, *Revisiting Radius, Diameter, and all Eccentricity Computation in Graphs through Certificates* 2018, hal-01729748.

Complexity aspects

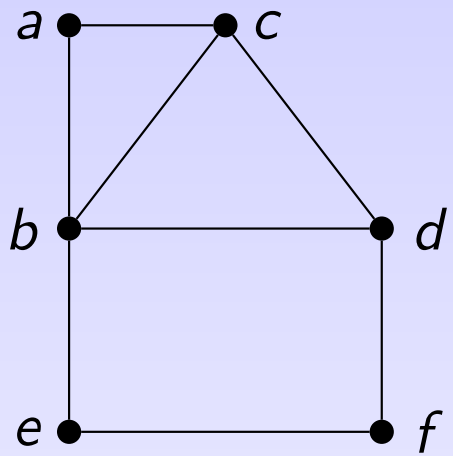
- Michele Borassi, Pierluigi Crescenzi, Michel Habib, *Into the Square : On the Complexity of Some Quadratic-time Solvable Problems*, **Electr. Notes Theor. Comput. Sci.** 322 : 51-67 (2016).

Algorithm 3: LBFS, Rose, Tarjan, and Lueker 1970s

Data: a graph $G = (V, E)$ and a start vertex s

Result: an ordering σ of V

Assign the label $\emptyset$ to all vertices
$label(s) \leftarrow \{n\}$
for $i \leftarrow n$ **à** 1 **do**
    Pick an unumbered vertex v with lexicographically largest label
    $\sigma(i) \leftarrow v$
    foreach *unnumbered vertex w adjacent to v* **do**
        $label(w) \leftarrow label(w).\{i\}$
    end
end



$\sigma(i)$	Affected Vertices	σ
$\sigma(6) = d$	$label(b) = label(c) = label(f) = 6$	d
$\sigma(5) = c$	$label(b) = 65$ and $label(a) = 5$	d, c
$\sigma(4) = b$	$label(a) = 54$ and $label(e) = 4$	d, c, b
$\sigma(3) = f$	$label(e) = 43$	d, c, b, f
$\sigma(2) = a$		d, c, b, f, a
$\sigma(1) = e$		d, c, b, f, a, e

FIGURE – A step by step computation of a LBFS ordering on G starting at vertex d.

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.

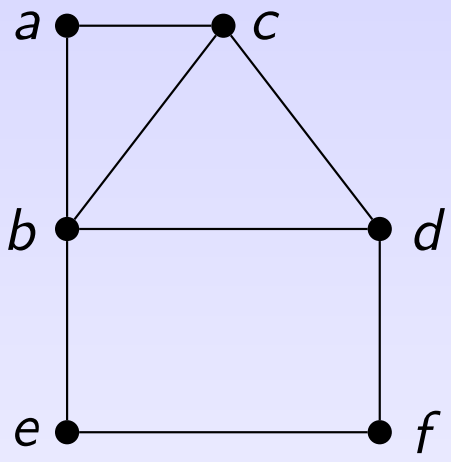


FIGURE – with
$\sigma_0 = d, c, b, f, a, e$

$$LBFS^+(G, \sigma_0) = \sigma_1 = e, f, b, d, c, a$$

Starting from and ordering of the vertices σ_0 we compute the following sequence : $\sigma_{i+1} = LBFS^+(G, \sigma_i)$.
It should be noticed that σ_{i+1} is uniquely defined by $LBFS^+(G, \sigma_i)$.

Starting from and ordering of the vertices σ_0 we compute the following sequence : $\sigma_{i+1} = LBFS^+(G, \sigma_i)$.
It should be noticed that σ_{i+1} is uniquely defined by $LBFS^+(G, \sigma_i)$.

Cycling

Due to the finite number of vertex orderings such a sequence must loop in a finite cycle of vertex orderings, which leads to the following definition.

Starting from and ordering of the vertices σ_0 we compute the following sequence : $\sigma_{i+1} = LBFS^+(G, \sigma_i)$.
It should be noticed that σ_{i+1} is uniquely defined by $LBFS^+(G, \sigma_i)$.

Cycling

Due to the finite number of vertex orderings such a sequence must loop in a finite cycle of vertex orderings, which leads to the following definition.

Definition

Let us define for a graph G, $LexCycle(G)$ as the **maximum** length of a cycle of vertex orderings obtained via a sequence of LBFS$^+$. [a]

a. It should be noticed that in this definition there is no assumption on the starting vertex ordering σ_0.

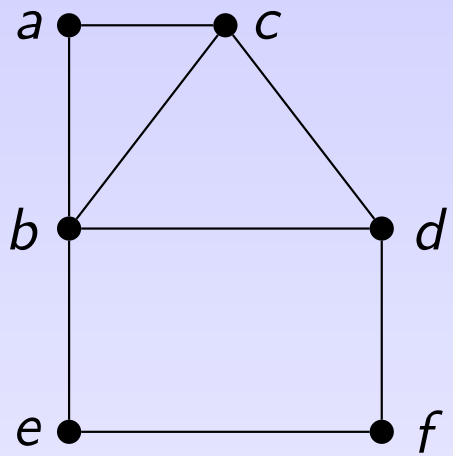


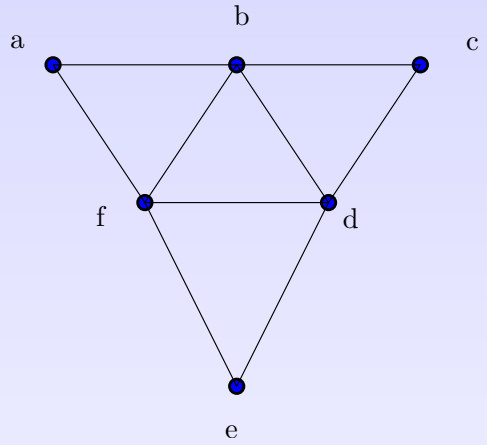
FIGURE – with
$\sigma_0 = d, c, b, f, a, e$

$$LBFS^+(G, \sigma_0) = \sigma_1 = e, f, b, d, c, a$$
$$LBFS^+(G, \sigma_1) = \sigma_2 = a, c, b, d, e, f$$
$$LBFS^+(G, \sigma_2) = \sigma_3 = f, e, b, d, c, a$$
$$LBFS^+(G, \sigma_3) = \sigma_4 = a, c, b, d, e, f = \sigma_2$$

Therefore with the cycle $[\sigma_2, \sigma_3, \sigma_2]$, $LexCycle(G) \geq 2$ and there is no cycle of length 3 (to be checked by hand) then

$$LexCycle(G) = 2.$$

Another example



$$\sigma_0 = f, d, a, e, b, c$$
$$\sigma_1 = LBFS^+(G, \sigma_0) = c, b, d, f, a, e$$
$$\sigma_2 = LBFS^+(G, \sigma_1) = e, f, d, b, a, c$$
$$\sigma_3 = LBFS^+(G, \sigma_2) = c, b, d, f, a, e = \sigma_1$$
$$C_2 = [\sigma_1, \sigma_2, \sigma_1]$$

- We will study here the first properties of this new graph invariant. Due to the $+$ rule, $\mathit{LexCycle}(G) \geq 2$.

- We will study here the first properties of this new graph invariant. Due to the $+$ rule, $\mathit{LexCycle}(G) \geq 2$.
- At first glance we know that $\mathit{LexCycle}(G) \leq |V(G)|!$, more precisely $\mathit{LexCycle}(G)$ is bounded by the number of LBFS orderings of G.

- We will study here the first properties of this new graph invariant. Due to the $+$ rule, $LexCycle(G) \geq 2$.
- At first glance we know that $LexCycle(G) \leq |V(G)|!$, more precisely $LexCycle(G)$ is bounded by the number of LBFS orderings of G.
- But there is no evidence for another general bound such as for example $|V(G)|^2$.

Asteroidal number

A set $A \subseteq V$ of G forms an asteroidal set if for each vertex $a \in A$, the set $A \backslash \{a\}$ is contained in one connected component of $G[V \backslash N[a]]$. The maximum cardinality of an asteroidal set of G, denoted $an(G)$, is called the *asteroidal number* of G. A graph is AT-free if it does not contain an asteroidal triple.

Asteroidal number

A set $A \subseteq V$ of G forms an asteroidal set if for each vertex $a \in A$, the set $A \backslash \{a\}$ is contained in one connected component of $G[V \backslash N[a]]$. The maximum cardinality of an asteroidal set of G, denoted $an(G)$, is called the *asteroidal number* of G. A graph is AT-free if it does not contain an asteroidal triple.

Conjecture Juraj Stacho (2015)

$LexCycle(G) \leq an(G)$

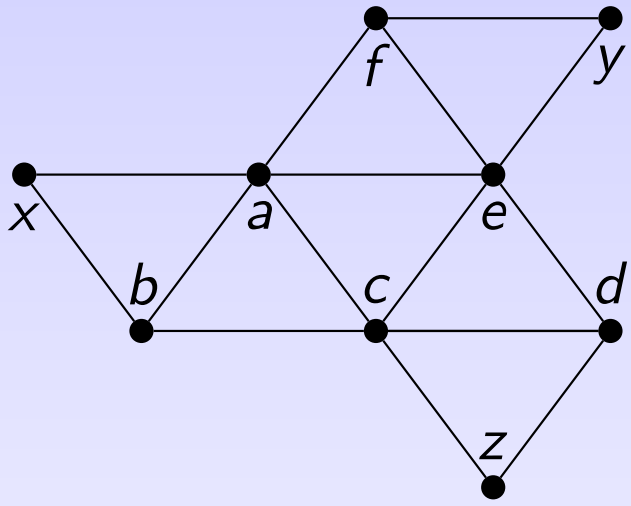
Asteroidal number

A set $A \subseteq V$ of G forms an asteroidal set if for each vertex $a \in A$, the set $A \backslash \{a\}$ is contained in one connected component of $G[V \backslash N[a]]$. The maximum cardinality of an asteroidal set of G, denoted $an(G)$, is called the *asteroidal number* of G. A graph is AT-free if it does not contain an asteroidal triple.

Conjecture Juraj Stacho (2015)

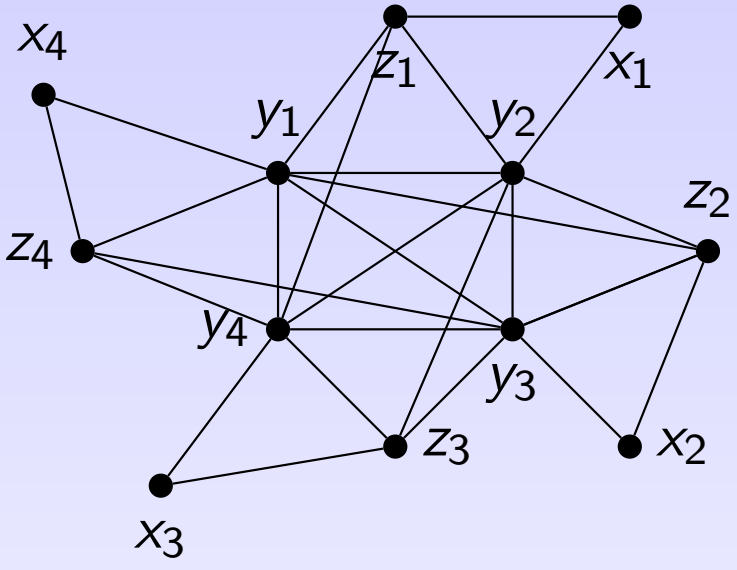
$LexCycle(G) \leq an(G)$

But unfortunately we will now construct a counterexample. Let us consider first some interesting examples with high values of *LexCycle*.



$$\sigma = \text{LBFS}(G_3) = x, b, a, c, e, f, d, z, y$$
$$\tau = \text{LBFS}^+(G_3, \sigma) = y, f, e, a, c, d, b, x, z$$
$$\theta = \text{LBFS}^+(G_3, \tau) = z, d, c, e, a, b, f, y, x$$
$$\sigma = \text{LBFS}^+(G_3, \theta) = x, b, a, c, e, f, d, z, y$$

FIGURE – G_3 with an asteroidal triple (x, y, z), and 3-cycle $C_3 = [\sigma, \tau, \theta, \sigma]$.



$$\sigma = \mathsf{LBFS}(G_4) = x_4, z_4, y_1, y_3, y_4, y_2, z_2, z_1, z_3, x_2, x_3, x_1$$
$$\tau = \mathsf{LBFS}^+(G_4, \sigma) = x_1, z_1, y_2, y_4, y_1, y_3, z_3, z_2, z_4, x_3, x_4, x_2$$
$$\theta = \mathsf{LBFS}^+(G_4, \tau) = x_2, z_2, y_3, y_1, y_2, y_4, z_4, z_3, z_1, x_4, x_1, x_3$$
$$\epsilon = \mathsf{LBFS}^+(G_4, \theta) = x_3, z_3, y_4, y_2, y_3, y_1, z_1, z_4, z_2, x_1, x_2, x_4$$
$$\sigma = \mathsf{LBFS}^+(G_4, \epsilon) = x_4, z_4, y_1, y_3, y_4, y_2, z_2, z_1, z_3, x_2, x_3, x_1$$
$$C_4 = [\sigma, \tau, \theta, \epsilon, \sigma]$$

Starjoin

For a family of graphs $\{G_i\}_{1 \leq i \leq k}$, we define $H = Starjoin(G_1, \dots G_k)$ as follows : For $i \in [k]$, add a universal vertex g_i to G_i, then add a root vertex r adjacent to all g_i's.

Starjoin

For a family of graphs $\{G_i\}_{1 \leq i \leq k}$, we define $H = Starjoin(G_1, \dots G_k)$ as follows : For $i \in [k]$, add a universal vertex g_i to G_i, then add a root vertex r adjacent to all g_i's.

Property

If $H = Starjoin(G_1, \dots G_k)$ then $an(H) = max\{k, max_{1 \leq i \leq k}\{an(G_i)\}\}$ and $LexCycle(H) \geq lcm_{1 \leq i \leq k}\{|C_i|\}$ where lcm stands for least common multiple, and C_i is a cycle in a sequence of LBFS$^+$ orderings of G_i.

Starjoin

For a family of graphs $\{G_i\}_{1 \leq i \leq k}$, we define $H = Starjoin(G_1, \dots G_k)$ as follows : For $i \in [k]$, add a universal vertex g_i to G_i, then add a root vertex r adjacent to all g_i's.

Property

If $H = Starjoin(G_1, \dots G_k)$ then $an(H) = max\{k, max_{1 \leq i \leq k}\{an(G_i)\}\}$ and $LexCycle(H) \geq lcm_{1 \leq i \leq k}\{|C_i|\}$ where lcm stands for least common multiple, and C_i is a cycle in a sequence of LBFS$^+$ orderings of G_i.

Counterexample

If $H = Starjoin(G_3, G_4)$, then $an(H) = 5$ and $LexCycle(H) \geq 12$.

REPEATED LBFS$^+$

Algorithm 4: LBFS$^+$ multi-sweep

Require: $G = (V, E)$
Ensure: an ordering σ
  $\sigma \leftarrow$ LBFS(G)
  for $i = 2$ **to** $|V|$ **do**
    $\sigma \leftarrow$ LBFS$^+$(G,σ)
  end for

Conjecture J. Dusart, M. Habib 2013

LexCycle(G)=2 for cocomparability graphs [a].

a. Could be extended to AT-free graphs

Conjecture J. Dusart, M. Habib 2013

LexCycle(G)=2 for cocomparability graphs [a].

a. Could be extended to AT-free graphs

Experimental results

Of course before asking this question we check on millions of cocomparability graphs (easy to generate).

Comparability graphs

Comparability graph

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

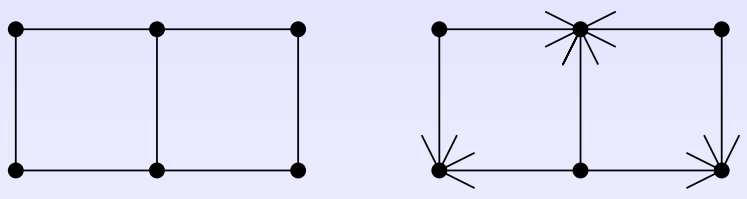


FIGURE – A comparability graph G and a transitive orientation of G.

Comparability graphs

Comparability graph

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

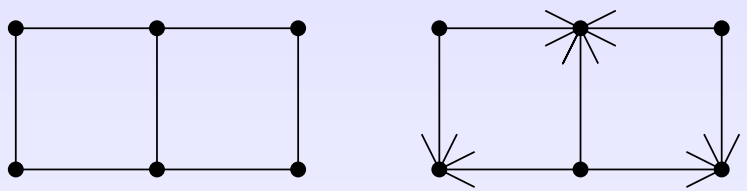


FIGURE – A comparability graph G and a transitive orientation of G.

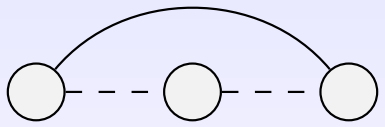
Cocomparability graph

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

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.

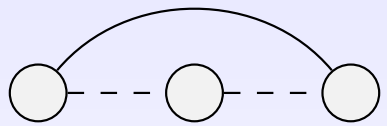


a, b, c, an umbrella

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

Remarks :

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

Vertex orderings

Vertex orderings are very useful for recognition algorithms, when a graph class can be defined by the existence of an ordering avoiding some patterns.

Vertex orderings

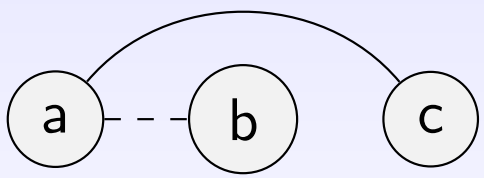
Vertex orderings are very useful for recognition algorithms, when a graph class can be defined by the existence of an ordering avoiding some patterns.

- A graph is a co-comparability graph iff it admits a cocomp ordering

Vertex orderings

Vertex orderings are very useful for recognition algorithms, when a graph class can be defined by the existence of an ordering avoiding some patterns.

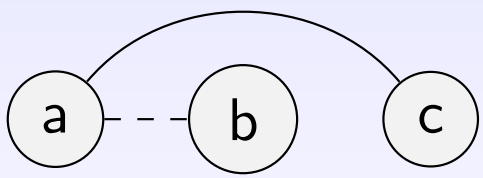
- A graph is a co-comparability graph iff it admits a cocomp ordering
- A graph is an interval graph iff it admits an interval ordering

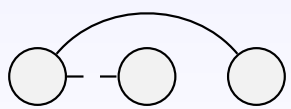
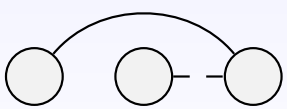


Vertex orderings

Vertex orderings are very useful for recognition algorithms, when a graph class can be defined by the existence of an ordering avoiding some patterns.

- A graph is a co-comparability graph iff it admits a cocomp ordering
- A graph is an interval graph iff it admits an interval ordering



- A graph is a proper interval graph iff it admits a proper interval ordering  and 

- Roughly speaking REPEATED LBFS$^+$ provides such an ordering for interval (resp. proper interval, cocomparability) graphs.

- Roughly speaking REPEATED LBFS$^+$ provides such an ordering for interval (resp. proper interval, cocomparability) graphs.
- Which can be used as a certificate in the positive case.

A very useful lemma (Corneil, Dusart, Habib, Kőhler 2011)

The flipping lemma

if σ is a cocomp ordering and $\tau = LBFS^+(G, \sigma)$, then for every non edge $xy \notin E(G)$
$x <_\sigma y$ iff $y <_\tau x$
In other words σ and τ^d are two linear extensions of the same transitive orientation of $\overline{G}$.

The particular case of interval graphs

Theorem (Corneil 2004) (Hell, Huang 2004)

For a proper interval graph, a series of 3 $LBFS^+$ produces a proper interval ordering.

The particular case of interval graphs

Theorem (Corneil 2004) (Hell, Huang 2004)

For a proper interval graph, a series of 3 $LBFS^+$ produces a proper interval ordering.

Theorem (Corneil, Olariu and Stewart 2010)

For an interval graph, a series of 5+1 special consecutive $LBFS^+$ produces an interval ordering.

Theorem (Dusart, Habib 2013)

If $G = (V, E)$ is a cocomparability graph REPEATED LBFS$^+$ always finds a cocomp ordering (so in less than $|V|$ LBFS$^+$).

Theorem (Dusart, Habib 2013)

If $G = (V, E)$ is a cocomparability graph REPEATED LBFS$^+$ always finds a cocomp ordering (so in less than $|V|$ 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.

Theorem (Dusart, Habib 2013)

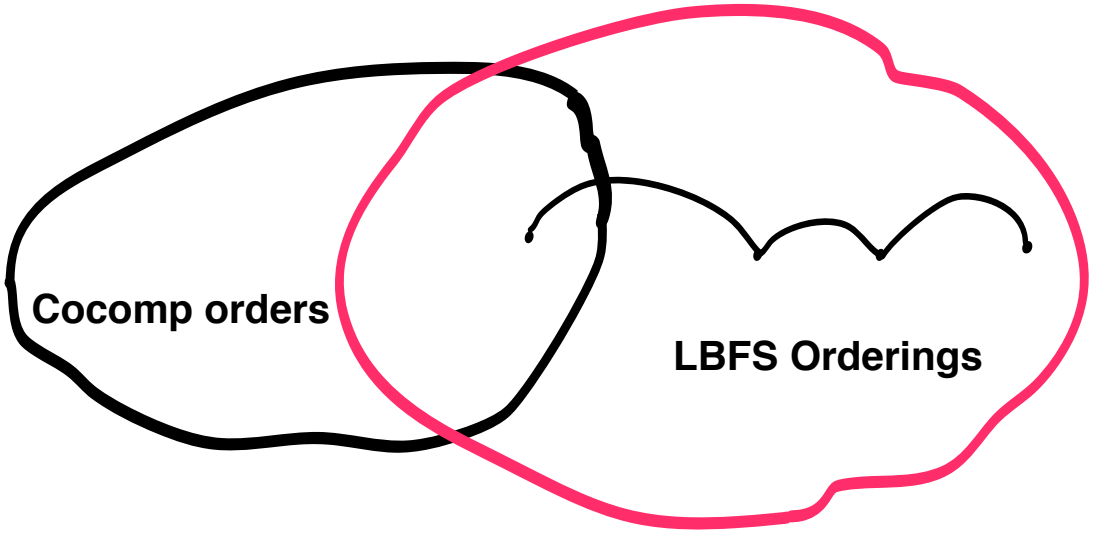
If $G = (V, E)$ is a cocomparability graph REPEATED LBFS$^+$ always finds a cocomp ordering (so in less than $|V|$ LBFS$^+$).

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.

Alea typical question

What is the average convergence of REPEATED LBFS$^+$?



Landscape for cocomparability graphs

Consequences

Dusart, Habib 2013

It gives a very easy to program : REPEATED LBFS$^+$ for cocomparability graph recognition or transitive orientation of a comparability graph with $O(nm)$ worst-case complexity.

Consequences

Dusart, Habib 2013

It gives a very easy to program : REPEATED LBFS$^+$ for cocomparability graph recognition or transitive orientation of a comparability graph with $O(nm)$ worst-case complexity.

If the LexCycle conjecture is true

We could avoid some LBFS$^+$ with the following algorithm.

Algorithm 5: Potential cocomp ordering or transitive orientation algorithm

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;

Partial results

- LexCycle(T)=2 for every tree T (in fact a result on BFS).

Partial results

- LexCycle(T)=2 for every tree T (in fact a result on BFS).
- For a given class of graphs $\mathcal{C}$, if for all prime graphs H $LexCycle(H) = 2$, then for every graph $G \in \mathcal{C}$ $LexCycle(G) = 2$. So the result holds for cographs.

Partial results

- LexCycle(T)=2 for every tree T (in fact a result on BFS).
- For a given class of graphs $\mathcal{C}$, if for all prime graphs H $LexCycle(H) = 2$, then for every graph $G \in \mathcal{C}$ $LexCycle(G) = 2$. So the result holds for cographs.
- If G is a proper interval then Algorithm 3 stops with $\sigma_5 = \sigma_3$ and $\sigma_4 = \sigma_3^d$.

Partial results

- LexCycle(T)=2 for every tree T (in fact a result on BFS).
- For a given class of graphs $\mathcal{C}$, if for all prime graphs H $LexCycle(H) = 2$, then for every graph $G \in \mathcal{C}$ $LexCycle(G) = 2$. So the result holds for cographs.
- If G is a proper interval then Algorithm 3 stops with $\sigma_5 = \sigma_3$ and $\sigma_4 = \sigma_3^d$.
- $LexCycle(G) = 2$ for interval graphs and cobipartite graphs and domino-free cocomparability graphs. But also splits graphs (not always cocomparability graphs).

Alea typical question

What is the average number of steps of LBFS^+ to reach a cycle ?

Minimum Path Cover for cocomp graphs with LDFS

- To find a Minimum Path Cover is NP-hard for arbitrary graphs (cf. Sophie Laplante's lecture)

Minimum Path Cover for cocomp graphs with LDFS

- To find a Minimum Path Cover is NP-hard for arbitrary graphs (cf. Sophie Laplante's lecture)
- Let P be a transitive orientation of $\overline{G}$
  our problem reduce to computing the bump number of P
  (Ugly polynomial algorithm MH, Möhring, Steiner 1988)

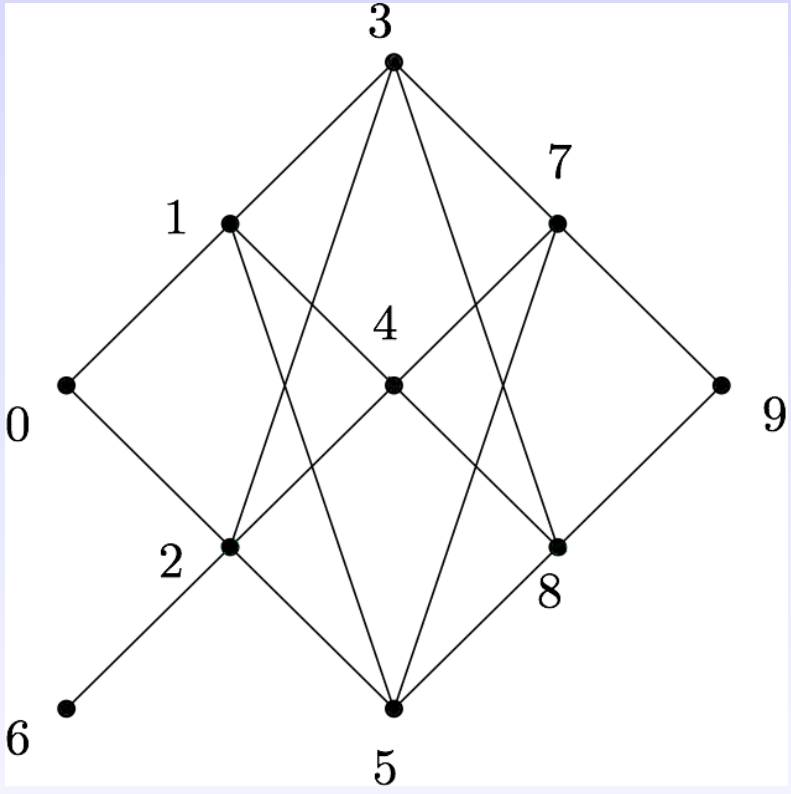
Minimum Path Cover for cocomp graphs with LDFS

- To find a Minimum Path Cover is NP-hard for arbitrary graphs (cf. Sophie Laplante's lecture)
- Let P be a transitive orientation of $\overline{G}$
  our problem reduce to computing the bump number of P
  (Ugly polynomial algorithm MH, Möhring, Steiner 1988)
- Another equivalent formulation as the 2-machine scheduling problem
  (Another polynomial algorithm Gabow, Tarjan 1985)

Minimum Path Cover

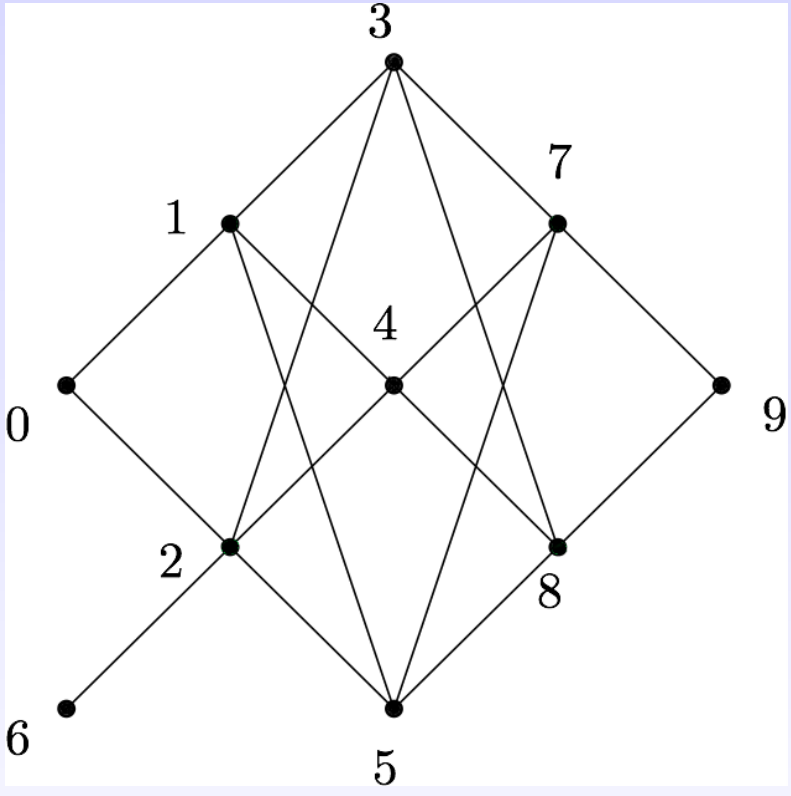
1. Start with σ any co-comparability ordering of G
2. Apply $LDFS^+(G, \sigma)$ to produce an ordering τ.
3. Apply $RightMostNeighbour(\tau)$ which gives the path cover
4. Exhibit a certificate of minimality with a cut-set.

Let us take an example



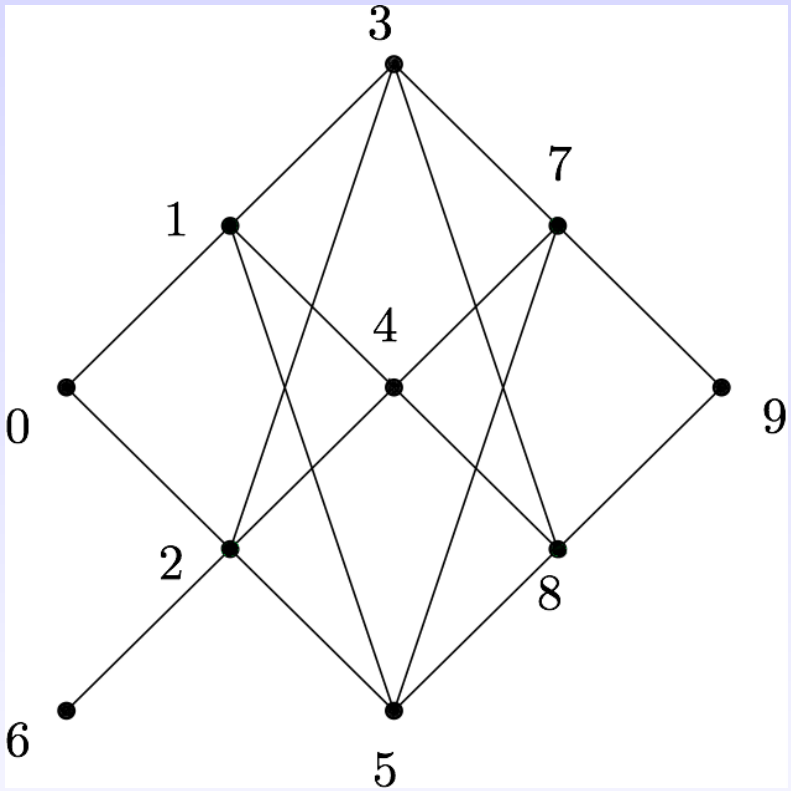
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering

Let us take an example



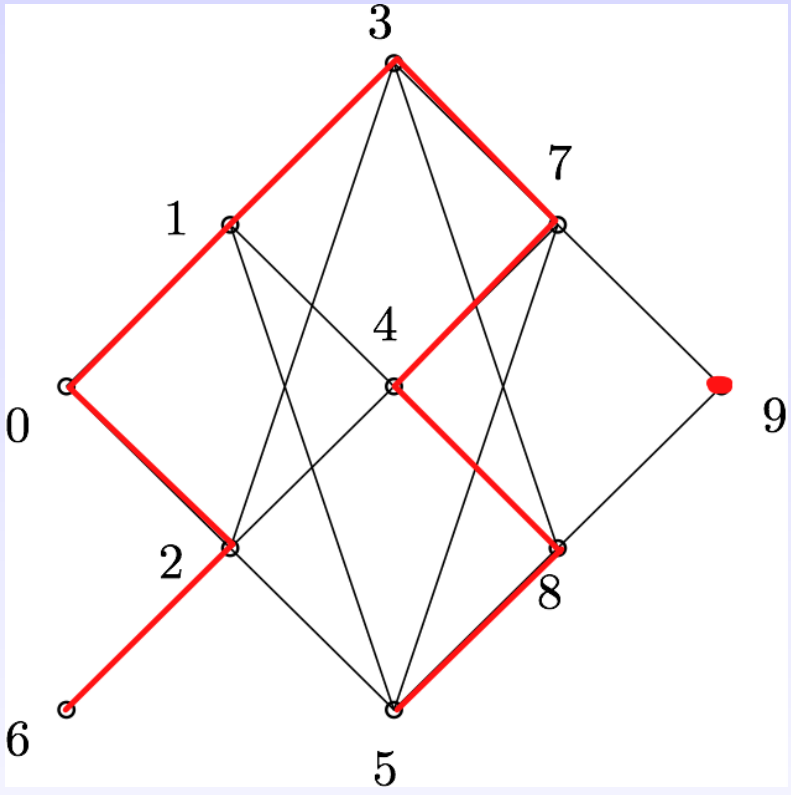
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering
2. $\tau = LDFS^{+}(G, \sigma) = 9, 8, 5, 7, 4, 1, 3, 2, 0, 6$

Let us take an example



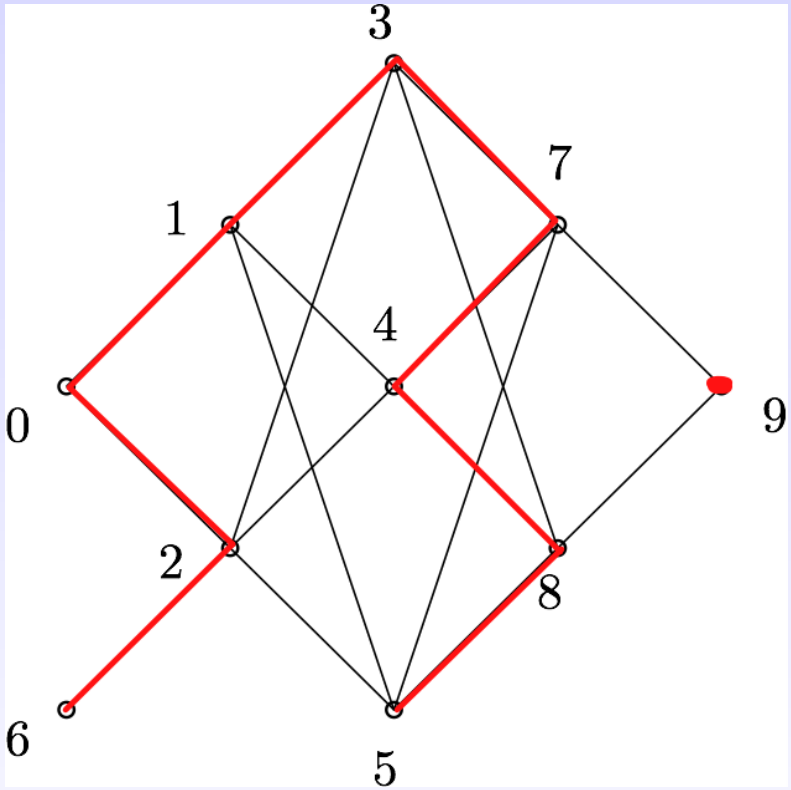
1. $\sigma = 2, 6, 0, 3, 4, 5, 1, 7, 8, 9$ a co-comparability ordering
2. $\tau = LDFS^{+}(G, \sigma) = 9, 8, 5, 7, 4, 1, 3, 2, 0, 6$
3. $RightMostNeighbour(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Magic



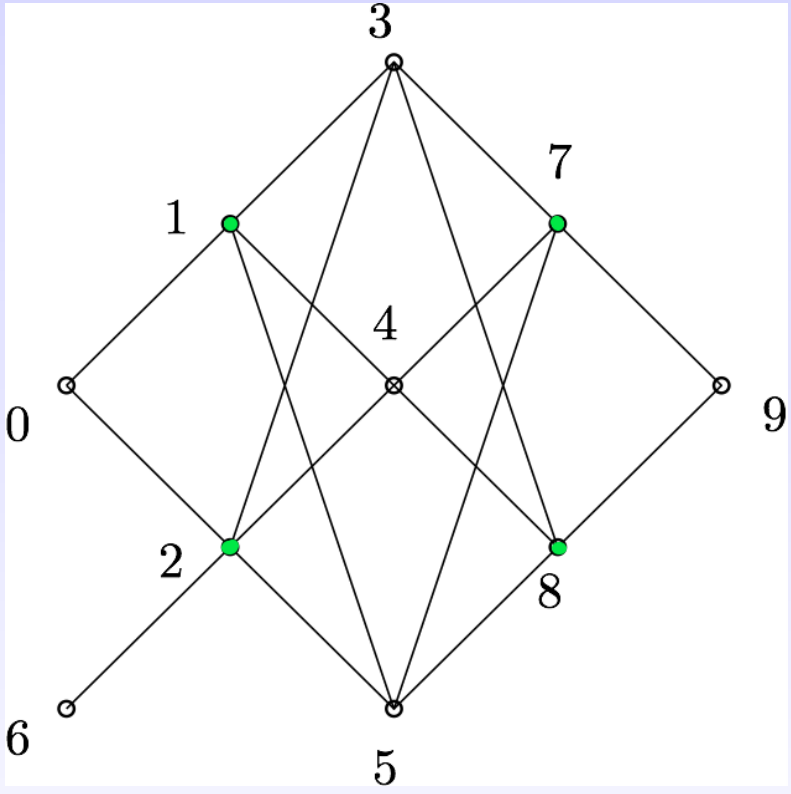
1. $\mathit{RightMostNeighbour}(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$

Magic



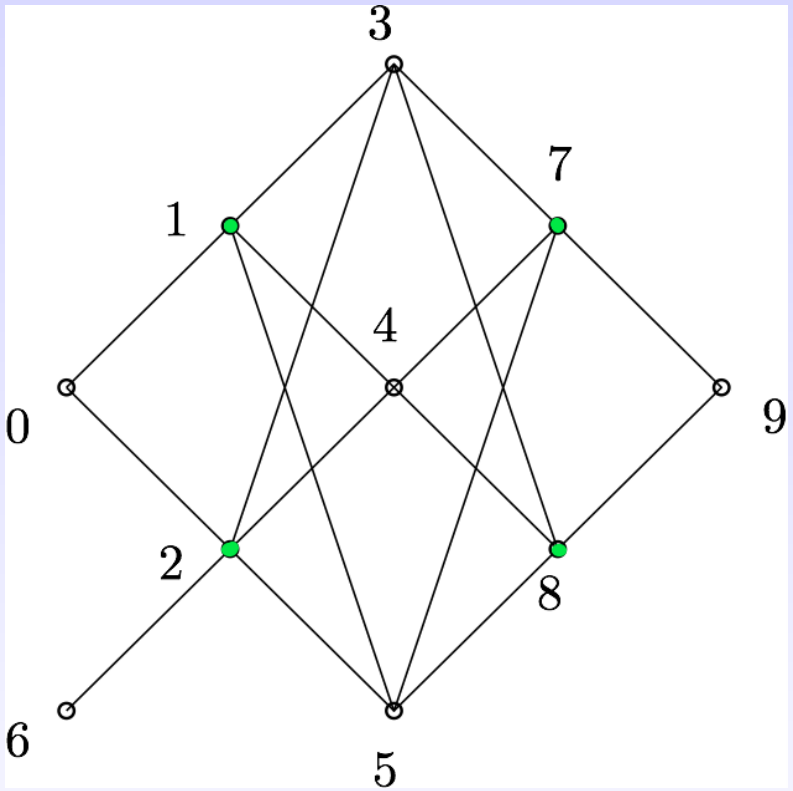
1. $\mathit{RightMostNeighbour}(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$
2. The cutset $S = \{1, 7, 2, 8\}$ disconnects G into 6 connected components.

Cutset



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

Cutset



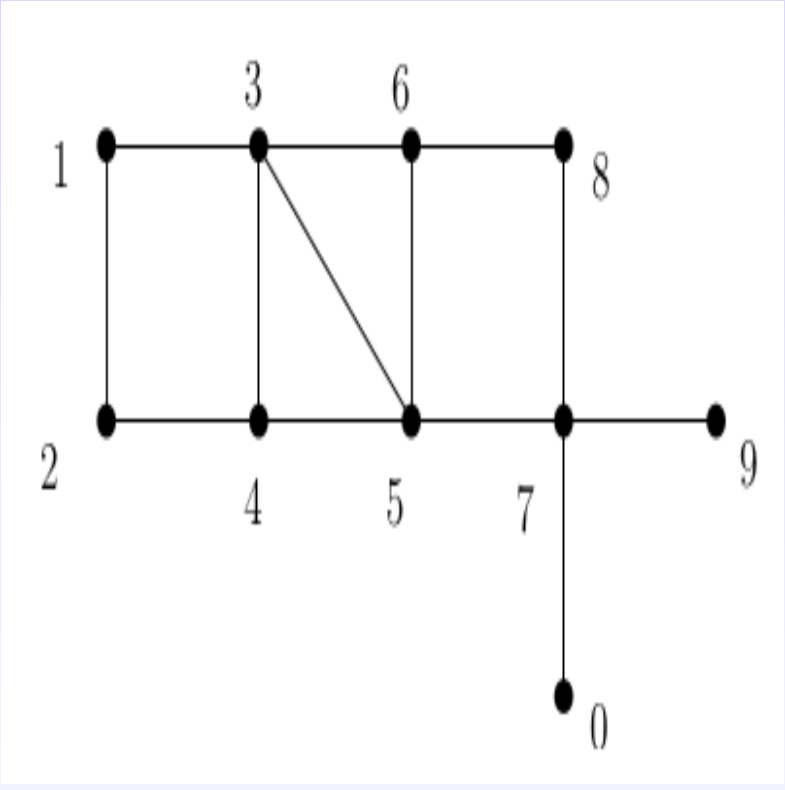
1. $\mathit{RightMostNeighbour}(\tau) = 6, 2, 0, 1, 3, 7, 4, 8, 5, ||9$
2. The cutset $S = \{1, 7, 2, 8\}$ disconnects G into 6 connected components. These vertices can be obtained during the rightmost neighbour procedure and correspond to backward

Although the algorithm is quite simple,
the proof is not that simple !
Since we need to prove the certifying step by induction.

Maximum independent set for cocomparability graph with LDFS

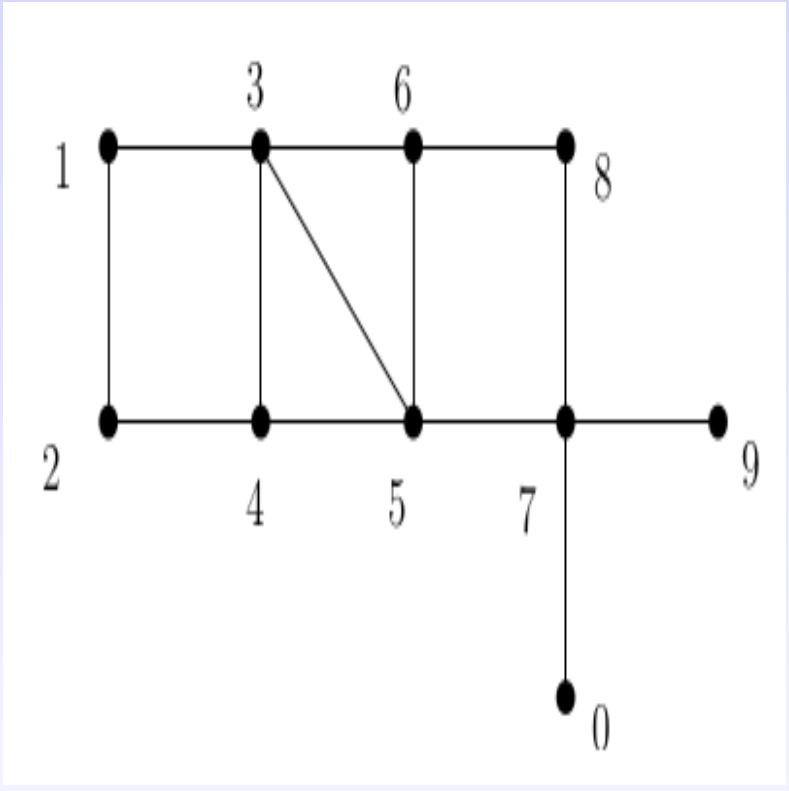
1. Start with σ any co-comparability ordering of G (a linear extension of P)
2. Apply $LDFS^+(G, \sigma)$ to produce an ordering τ.
3. Apply $GreedyIndependentSet(\tau)$ which gives the independent set
4. Exhibit a certificate of minimality with a clique cover.

Let us take an example



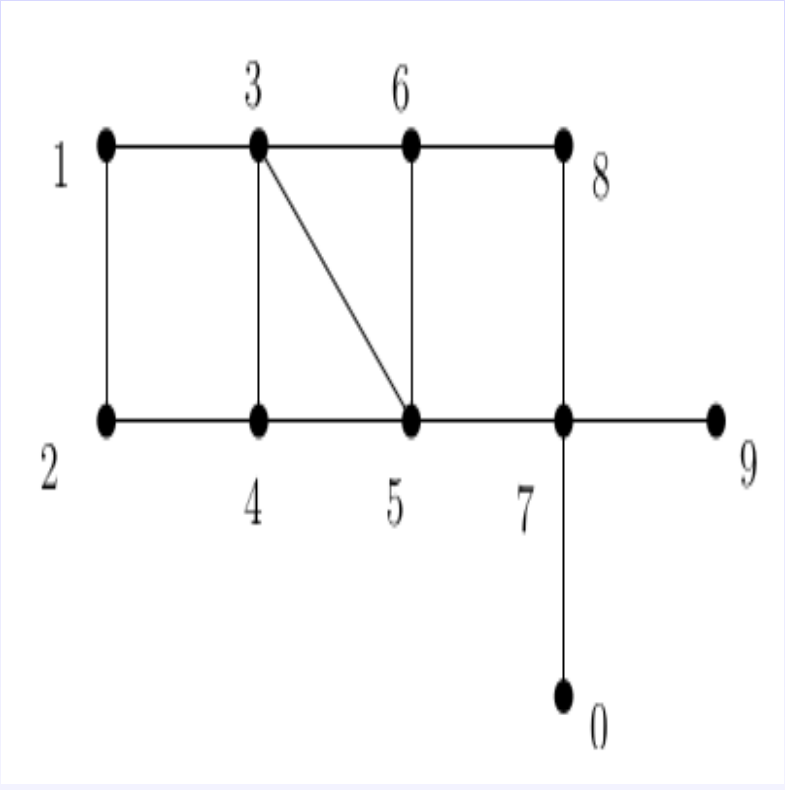
1. $\sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$ a co-comparability ordering

Let us take an example



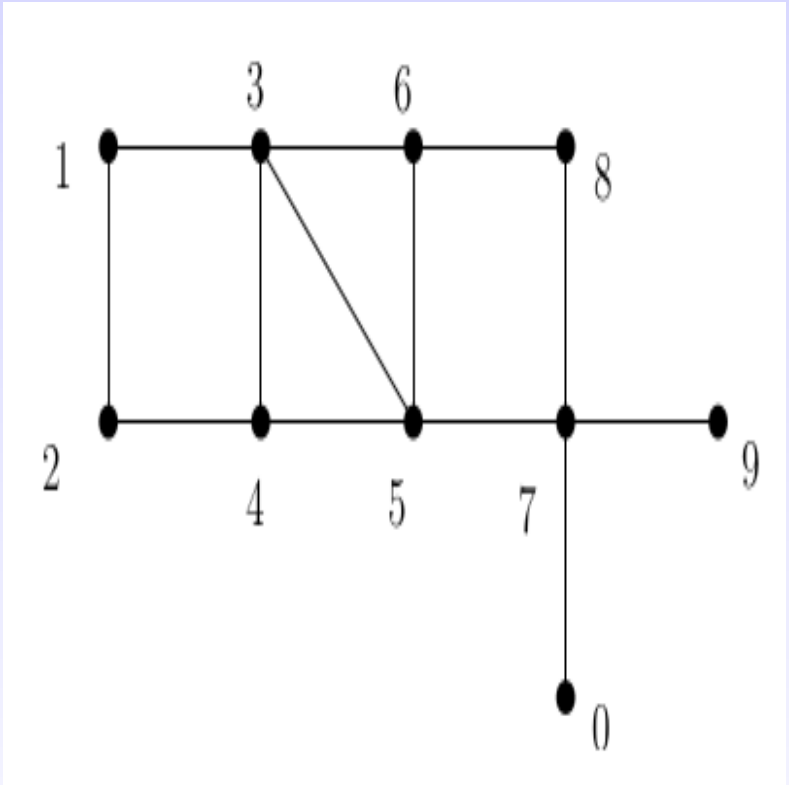
1. $\sigma = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$ a co-comparability ordering
2. $\tau = LDFS^{+}(G, \sigma) = 0, 7, 9, 8, 6, 5, 3, 4, 2, 1$

Let us take an example



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

Let us take an example



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

- Applying a series of graph searches can be done even on huge graphs (cf. BFSs for diameter computations)
- Easy extensions : to weighted graphs by substituting Dijkstra's algorithm for BFS and to directed graphs
- How much can we learn about the structure of a graph with a series of graph searches ?
- Other graph searches such as DFS, LDFS ... could also have such cycling properties with the $+$ rule.
- Understand the "matroidal" aspects of LDFS. We have found around 10 algorithms that can be put in this framework (including the Kosaraju and Sharir's algorithm for strongly connected components), in which the lexicographic rightmost object is the optimum.

Many thanks for your attention ! !