Using Simple BFS or DFS for some Graph Classes Recognition

Binh-Minh Bui-Xuan (Lip6)
Michel Habib (IRIF)
Fabien de Montgolfier (IRIF)
Renaud Torfs (IRIF)

Friday June 5th 2026

Graph searches

Unit Interval Graphs (UIG)

Trivially Perfect Graphs

Conclusion

Graph searching

Algorithm

- Mark the root as *seen*
- While unvisited and seen vertices remain:
  - pick and visit a vertex x
  - mark its unseen neighbors as seen
  - $ordering[i++] \leftarrow x$

States

- Visited
- Unvisited
  - Seen
  - Unseen

Tie-break rules

BFS (Breadth-First Search)

Visit the longest-waiting seen vertex

- uses a FIFO queue

DFS (Depth-First Search)

Visit the shortest-waiting seen vertex

- Uses the recursion stack

LexBFS

Lexicographic Breadth-First Search (LexBFS)

- Vertices have labels
- When visiting x, for a neighbor $Label(y) \leftarrow Label(y).x$
- Visit vertex with largest label (wrt lexicographic order)

Linear-time implementation

- Rose, Tarjan and Lueker [1976]
- Habib, Paul and Viennot [1996]

LexBFS

Why?

- Chordal graph recognition
- Maximum Clique for chordal graphs
- Proper coloring for Distance-hereditary graphs
- Interval Graph recognition
- Cographs recognition
- Proper Interval Graphs recognition
- Trivially Perfect Graphs recognition
- ...

Going further

Upgrading LexBFS

- Maximum Cardinality Search (MCS)
- Maximum Neighborhood Search (MNS)
- LexDFS
- $LexBFS^{+}$
- $LexBFS^{*}$ etc.

Going further

Upgrading LexBFS

- Maximum Cardinality Search (MCS)
- Maximum Neighborhood Search (MNS)
- LexDFS
- $LexBFS^+$
- $LexBFS^*$ etc.

Downgrading LexBFS

Use simpler graph searches for the same purpose

Degree-guided graph searching

Four degree-guided graph searches

- minBFS
- maxBFS
- minDFS
- maxDFS

Rule

Among seen vertices, visit the one with minimum (resp. maximum) degree

How difficult?

Use standard, existing BFS or DFS implementation

1. Sort all adjacency list in decreasing (resp. increasing) degree
2. perform BFS (or DFS)

How to sort all adjacency lists of G wrt σ?

- new edgeless graph G_s with same vertex set than G
- for every vertex u of G taken in σ order
  - for every neighbor v of u
    - add u at end of v adjacency list in G_s

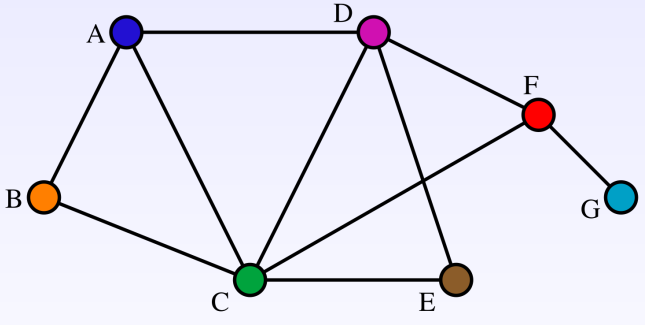
Definition

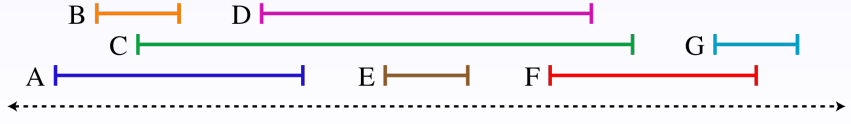
Realizer

A collection of n intervals on the real line

Interval Graph (of a realizer)

- On vertex per interval
- Edges = intersections





Definition

Proper Interval Graph

No interval is included in another one

Unit Interval Graph

All intervals have length 1

Nested Interval Graph

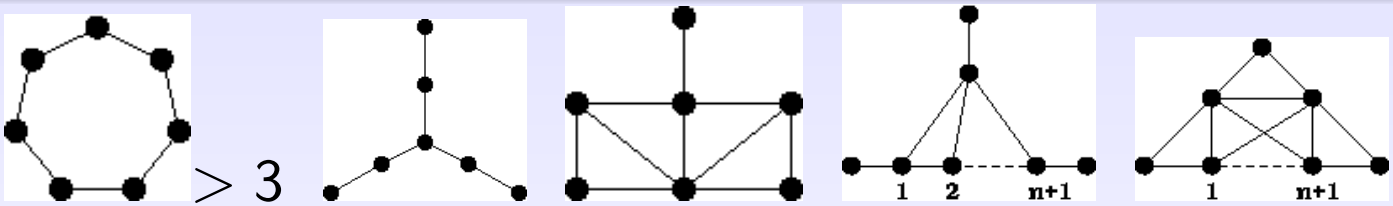
No overlap: either two intervals are disjoint; or one contains the other

Interval Graph Recognition

Recognition problem

Given a graph G, is it an Interval Graph ? Bonuses:

- If yes, output realizer.
- If no, output obstruction.



Linear-time algorithms

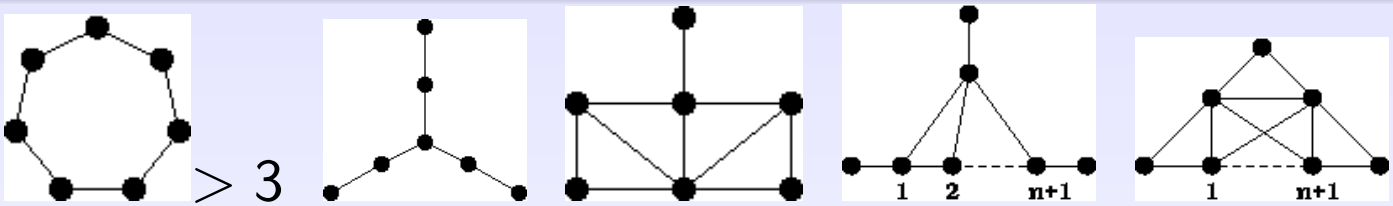
- Booth and Lueker [1976] PQ-tree
- Corneil, Olariu, Stewart [SODA 1998] *"The ultimate interval graph recognition algorithm?"* 4 LexBFS

Interval Graph Recognition

Recognition problem

Given a graph G, is it an Interval Graph ? Bonuses:

- If yes, output realizer.
- If no, output obstruction.



Linear-time algorithms

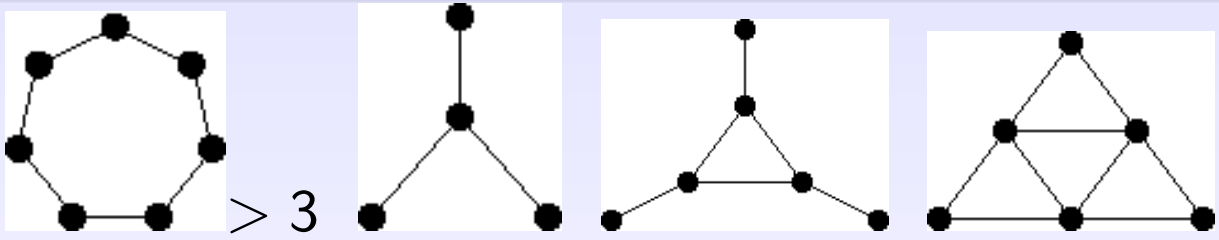
- Booth and Lueker [1976] PQ-tree
- Corneil, Olariu, Stewart [SODA 1998] *"The ultimate interval graph recognition algorithm?"* 4 LexBFS
- Corneil, Olariu, Stewart [2009] : 6 LexBFS. Conjecture: 5.
- Li and Wu [2014] 4 LexBFS

Unit Interval Graph Recognition

Recognition problem

Given a graph G, is it an Unit Interval Graph ? Bonuses:

- If yes, output realizer.
- If no, output obstruction.



Linear-time algorithms

- Loodge and Olariu [1993]
- Deng, Hell, and Huang [1996] direct construction
- de Figueiredo, Meidanis, de Mello [1995race condition] LexBFS
- Corneil [2004] *"A simple 3-sweep LexBFS..."*

Unit Interval Graph characterization

Theorem: the following are equivalent

1. G is an Unit Interval Graph (UIG)
2. G is a Proper Interval Graph (PIG)
3. G admits an Indifference Ordering
4. G admits a Consecutive Neighborhood Ordering
5. G admits a LO Ordering
6. G admits a Bisimplicial Elimination Ordering

1 Unit Interval Graph

All intervals of the realizer have length 1

Unit Interval Graph characterization

Theorem: the following are equivalent

1. G is an Unit Interval Graph (UIG)
2. G is a Proper Interval Graph (PIG)
3. G admits an Indifference Ordering
4. G admits a Consecutive Neighborhood Ordering
5. G admits a LO Ordering
6. G admits a Bisimplicial Elimination Ordering

2 Proper Interval Graph

No interval of the realizer is included in another one

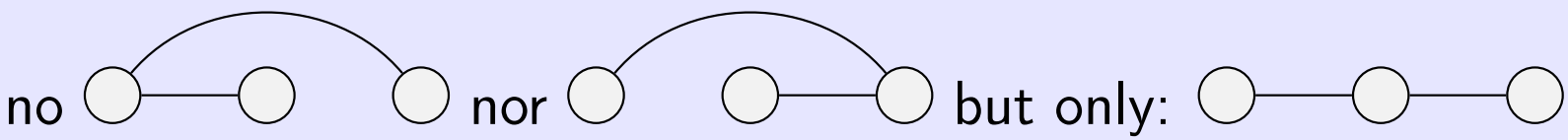
Unit Interval Graph characterization

Theorem: the following are equivalent

1. G is an Unit Interval Graph (UIG)
2. G is a Proper Interval Graph (PIG)
3. G admits an Indifference Ordering
4. G admits a Consecutive Neighborhood Ordering
5. G admits a LO Ordering
6. G admits a Bisimplicial Elimination Ordering

3 Indifference Ordering

Total ordering of the vertices such that for every induced P_3



Unit Interval Graph characterization

Theorem: the following are equivalent

1. G is an Unit Interval Graph (UIG)
2. G is a Proper Interval Graph (PIG)
3. G admits an Indifference Ordering
4. G admits a Consecutive Neighborhood Ordering
5. G admits a LO Ordering
6. G admits a Bisimplicial Elimination Ordering

4 Consecutive Neighborhood Ordering

Total ordering such that, for any vertex v, its closed neighborhood $N[v]$ occurs consecutively in that ordering

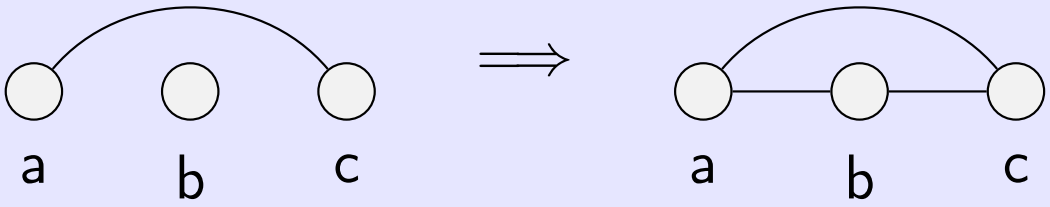
Unit Interval Graph characterization

Theorem: the following are equivalent

1. G is an Unit Interval Graph (UIG)
2. G is a Proper Interval Graph (PIG)
3. G admits an Indifference Ordering
4. G admits a Consecutive Neighborhood Ordering
5. G admits a LO Ordering
6. G admits a Bisimplicial Elimination Ordering

5 LO Ordering (LO stands for Loodge and Olariu [1993])

Total ordering such that, for every $a < b < c$,



Unit Interval Graph characterization

Theorem: the following are equivalent

1. G is an Unit Interval Graph (UIG)
2. G is a Proper Interval Graph (PIG)
3. G admits an Indifference Ordering
4. G admits a Consecutive Neighborhood Ordering
5. G admits a LO Ordering
6. G admits a Bisimplicial Elimination Ordering

6 Bisimplicial Elimination Ordr.: simplicial in both direction

Simplicial Elimination Ordering: for any vertex v, the neighbors of V that precede it in that ordering form a clique

So we have 6 different ordering...

Our starting question

Difference between the Left Bound Ordering of the realizer and the 4 other orderings? How many realizers and orderings may exist?

So we have 6 different ordering...

Our starting question

Difference between the Left Bound Ordering of the realizer and the 4 other orderings? How many realizers and orderings may exist?

Twins

Vertices that have the same neighborhood

Prime graph

Has no nontrivial module

Observation

An UIG is Prime $\iff$ it is connected and twin-free

One ordering to rule them all

Theorem: Let G be a prime UIG graph. The following orderings are unique and are the same, up to mirror:

1. the Left Bound ordering of a Unit realizer of G,
2. the Left Bound ordering of a Proper realizer of G,
3. an Indifference Ordering of G,
4. a Consecutive Neighborhood Ordering of G,
5. an LO Ordering of G,
6. a Bisimplicial Elimination Ordering of G,

One ordering to rule them all

Theorem: Let G be a prime UIG graph. The following orderings are unique and are the same, up to mirror:

1. the Left Bound ordering of a Unit realizer of G,
2. the Left Bound ordering of a Proper realizer of G,
3. an Indifference Ordering of G,
4. a Consecutive Neighborhood Ordering of G,
5. an LO Ordering of G,
6. a Bisimplicial Elimination Ordering of G,
7. `minBFS`(r) where r is an extremity of G.

Extremity

Leftmost or rightmost extremity of the realizer

One ordering to rule them all

Theorem: Let G be a prime UIG graph. The following orderings are unique and are the same, up to mirror:

1. the Left Bound ordering of a Unit realizer of G,
2. the Left Bound ordering of a Proper realizer of G,
3. an Indifference Ordering of G,
4. a Consecutive Neighborhood Ordering of G,
5. an LO Ordering of G,
6. a Bisimplicial Elimination Ordering of G,
7. `minBFS`(r) where r is an extremity of G.
8. `minDFS`(r) where r is an extremity of G.

Extremity

Leftmost or rightmost extremity of the realizer

LexBFS(UIG): a sledgehammer to crack a nut

Corneil uses LexBFS to recognize UIGs and compute these orderings. However...

LexBFS(UIG) breaks only one class

When visiting v, *i.e.*, refining with pivot-set $N(v)$

- All *seen* classes (non-empty label) are neighbors of v!
  - Label updated from L to $L.v$
- Only the *unseen* vertices class (empty label) may be split

LexBFS classes labels are intervals

$(j...i)$; $(j-1...i)$; ... ; $(i-1,i)$; (i) ; $\emptyset$

Lexicographic comparison resolves to length comparison

Given two unvisited vertices x and y, $label(x) > label(y)$
$\iff$ x has more visited neighbors than y

Proof of minBFS as UIG recognition algorithm

Invariant at step i

- The *seen* vertices form a clique
- If x is a seen vertex, $\exists j \leq i$ s.t. the visited neighbors of x are exactly $[j, ...i]$

Comparison lemma

Given two *seen* vertices x and y, if x has a private neighbor u and y a private neighbor v, then either u is visited and v unseen, or v is visited and u unseen.

Consequence

Among the eligible vertices of BFS, it is enough to take the one with *minimum degree*. dynamic comparison on labels resolves on static comparison on degree

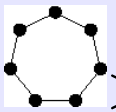
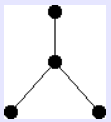
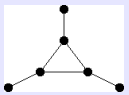
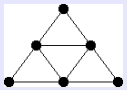
Conclusion for UIG

A two BFS algorithm

1. Find an extremity of the realizer: maxBFS
2. Compute the ordering and the realizer: minBFS or minDFS

Theorem

It can be tested in linear time if is an UIG

- The obstruction ( > 3 or  or  or ) may easily be output if it is not

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.
3. [Yan, Chen, Chang 1996: quasi-threshold graphs] a graph that can be constructed recursively by
   - add one universal vertex
   - disjoint union

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.
3. [Yan, Chen, Chang 1996: quasi-threshold graphs] a graph that can be constructed recursively by
   - add one universal vertex
   - disjoint union
4. Nested Interval graph

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.
3. [Yan, Chen, Chang 1996: quasi-threshold graphs] a graph that can be constructed recursively by
   - add one universal vertex
   - disjoint union
4. Nested Interval graph
5. a cograph that is also a chordal graph.

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.
3. [Yan, Chen, Chang 1996: quasi-threshold graphs] a graph that can be constructed recursively by
   - add one universal vertex
   - disjoint union
4. Nested Interval graph
5. a cograph that is also a chordal graph.
6. a cograph that is also an interval graph.

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.
3. [Yan, Chen, Chang 1996: quasi-threshold graphs] a graph that can be constructed recursively by
   - add one universal vertex
   - disjoint union
4. Nested Interval graph
5. a cograph that is also a chordal graph.
6. a cograph that is also an interval graph.
7. (C_4, P_4)-free graph.

Definition + Theorem : the following are equivalent

1. [Wolk 1962] comparability graph of a rooted forest
2. [Golumbic 1978] for every induced subgraph, size of a maximum independent set = number of maximal cliques.
3. [Yan, Chen, Chang 1996: quasi-threshold graphs] a graph that can be constructed recursively by
   - add one universal vertex
   - disjoint union
4. Nested Interval graph
5. a cograph that is also a chordal graph.
6. a cograph that is also an interval graph.
7. (C_4, P_4)-free graph.
8. a graph that admit a vertex ordering avoiding

Recognition of Trivially Perfect graphs

Yan, Chen, Chang [1996]

Claim linear time, but no complexity proof, and no certificate. Uses DFS as a tool to compute some degrees.

Frank Pok Man Chu [2008]

A simple patch over LexBFS. Produces positive and negative certificates.

Recognition using maxDFS

Theorem

maxDFS allows to recognize Trivially Perfect graphs in linear time

Positive certificate

Let G be Trivially Perfect. $maxDFS(G)$ yields an ordering avoiding

Negative certificate

Let G be *not* Trivially Perfect. maxDFS finds a C_4 or a P_4 in G.

Sketch of proof

- Triv perfect $\iff$ every component is Triv perfect

Sketch of proof

- Triv perfect $\iff$ every component is Triv perfect
- connected Trivially Perfect $\iff$ Comparability graph of a tree

Sketch of proof

- Triv perfect $\iff$ every component is Triv perfect
- connected Trivially Perfect $\iff$ Comparability graph of a tree
- Root is an universal vertex
- maxDFS starts at this root (maximum degree)

Sketch of proof

- Triv perfect $\iff$ every component is Triv perfect
- connected Trivially Perfect $\iff$ Comparability graph of a tree
- Root is an universal vertex
- maxDFS starts at this root (maximum degree)
- Removal of this roots disconnects.
- DFS explores components one by one

Sketch of proof

- Triv perfect $\iff$ every component is Triv perfect
- connected Trivially Perfect $\iff$ Comparability graph of a tree
- Root is an universal vertex
- maxDFS starts at this root (maximum degree)
- Removal of this roots disconnects.
- DFS explores components one by one
- max degree rule $\rightarrow$ start at new root
- The maxDFS tree is the underlying tree

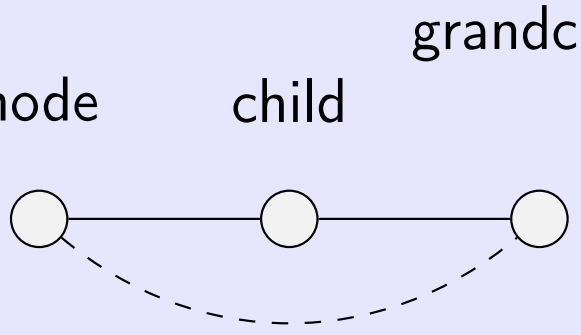
Sketch of proof

- Triv perfect $\iff$ every component is Triv perfect
- connected Trivially Perfect $\iff$ Comparability graph of a tree
- Root is an universal vertex
- maxDFS starts at this root (maximum degree)
- Removal of this roots disconnects.
- DFS explores components one by one
- max degree rule $\rightarrow$ start at new root
- The maxDFS tree is the underlying tree
- Neighborhood of a vertex = ancestors and descendants in DFS tree

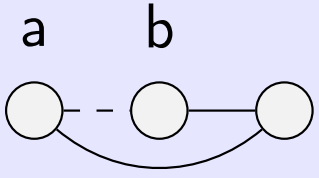
Sketch of proof

(*) Neighborhood of x = ancestors and descendants in DFS tree

Positive certificate



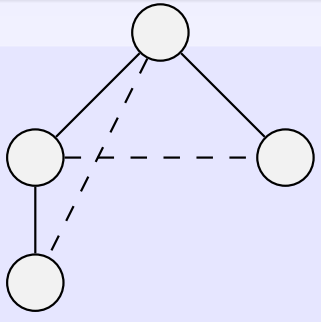
contradicts (*)



path from a to b : not chordal

Negative certificate

Check no edges between unrelated vertices



C_4 or P_4

Conclusion

For two graph classes where LexBFS was the best known recognition algorithm, we show sorting adjacency lists and then performing BFS or DFS is enough

And also

For chordal graphs BFS is enough... but with a *dynamic* condition on degree $\rightarrow$ needs using a heap $\rightarrow$ not linear

Perspectives

- Use simple searches to recognise other classes
- Relationships between orderings and searches