Graph Searching & Perfect Graphs*

Lalla Mouatadid
University of Toronto

Abstract

Perfect graphs, by definition, have a nice structure, that graph searching seems to extract in a, often non-inexpensive, manner. We scratch the surface of this elegant research area by giving two examples: Lexicographic Breadth Search on Chordal Graphs, and Lexicographic Depth First Search on Cocomparability graphs.

1 Introduction

Here's one particular way to cope with NP-hardness: Suppose we know some structure about the object we are dealing with, can we exploit said structure to come up with simple (hmm...?), efficient algorithms to NP-hard problems. One example of this structure is the "interval representation" most people are familiar with, where in such a setting we were able to solve the independent set problem in linear time, whereas the independent set problem on arbitrary graphs cannot even be approximated in polynomial time to a constant factor (unless P = NP).

Before we delve into the topic, let's recall some definitions.

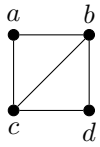
Let $G(V, E)$ be a **finite** and **simple** (no loops and no multiple edges) graph:

- The neighbourhood of a vertex v is the set of vertices adjacent to v. We write $N(v) = \{u | uv \in E\}$. The closed neighbourhood of v is $N[v] = N(v) \cup \{v\}$.
- The complement of G, denoted $\overline{G}(V, \overline{E})$ is the graph on the same set of vertices as G, where for all $u, v \in V, uv \in E \iff uv \notin \overline{E}$.
- A induced subgraph of G is a graph $G'(V', E')$ where $V' \subseteq V$ and $\forall u, v \in V'$, $uv \in E' \iff uv \in E$.
- A cycle $\mathcal{C}_k = v_1, v_2, \ldots, v_k$ in G is an induced subgraph where for $i \in [k], v_i v_{i+1 \mod k}$ are the only edges present in $\mathcal{C}_k$.
- A clique is a set $S \subseteq V$ where $\forall u, v \in S, uv \in E$.
- The clique number of G, denoted $\omega(G)$, is the size of the largest clique in G.
- An independent set is a set $S \subseteq V$ where $\forall u, v \in S, uv \notin E$.
- The independent set number of G, denoted $\alpha(G)$, is the size of the largest independent set in G.

*These notes are of a guest lecture I gave for the Advanced Algorithm Design course.

- A *proper* colouring is a function $col : V \to \{1, 2, \ldots, k\}$ that assigns values $i \in [k]$ such that $\forall uv \in E, col(u) \neq col(v)$.
- The chromatic number of G, denoted $\chi(G)$, is the minimum number of colours needed to properly colour G.
- A clique cover of G, is a partioning $P_1, P_2, \ldots, P_k$ of the vertices of V where $\bigcup_{i=1}^{k} P_i = V$ and $\forall i \in [k]$, P_i is a clique.
- The minimum clique number of G, denoted $\kappa(G)$, is the minimum number of cliques needed to cover G.
- A hole is an odd cycle on 5 or more vertices.
- An antihole is the complement of a hole.
- A graph G is *complete* if G is a clique.
- A vertex $v \in V$ is **simplicial** if $N(v)$ induces a clique.
- A *separator* of a graph G is a subset of vertices $S \subseteq V$ whose removal from G disconnects the graph into two or more connected components. An ab-separator is a separator of G that disconnects a from b. A ab-separator S is *minimal* if no proper subset of S also separates a from b.

Some (obvious) remarks: The complement of a clique is an independent set and vice versa. The clique number of a graph is always a lower bound to its chromatic number; and the clique cover number is always an upper bound to the indpendent set number. To illustrate the definitions above, consider the graph below:



$$\chi(G) = \omega(G) = 3$$
$$\alpha(G) = \kappa(G) = 2$$

The subgraph $H(V', E')$ where $V' = \{a, b, c\}$ and $E' = \{ab, ac\}$ is **not** an induced subgraph of G, since the edge $cb \notin E'$.

The set $S = \{a, b, c\}$ is a clique in G, $S' = \{a, b, d\}$ is not. The set $\{a, d\}$ is an independent set, $\{a, d, b\}$ is not.

A proper colouring of G would be $col(a) = col(d) = 1, col(b) = 2, col(c) = 3$, therefore $\chi(G) \leq 3$. Since $\omega(G) \leq \chi(G)$ and $\omega(G) = 3$, it follows $\chi(G) = 3$ is optimal.

Perfect Graphs: A graph family that received significant attention because of its nice structure is the class of *perfect graphs*.

Definition 1. *A graph $G(V, E)$ is* perfect *if for every induced subgraph H of G:*

$$\chi(H) = \omega(H)$$

Perfect graphs were introduced by Claude Berge in the sixties, and have since been well studied. In fact, many NP-hard problems can be solved efficiently on perfect graphs using the ellipsoids method, however research is still being developed to come up with truly combinatorial algorithms.

Many graph families belong to the class of perfect graphs; interval graphs for instance, permutation graphs - which you may have seen in the longest subsequence problem. We will focus on a different graph class, known as *chordal graphs*. First, we list two main theorems regarding perfect graphs.

Theorem 1 (The Weak Perfect Graph Theorem)**.** *A graph is perfect if and only if its complement is perfect.*

Theorem 2 (The Strong Perfect Graph Theorem)**.** *A graph is perfect if and only if it does not contain an induced hole or an induced antihole.*

Graph Searching:

Definition 2. *Graph searching is a mechanism to traverse the graph one vertex at a time in a specific manner.*

Classical graph searches you've seen before are BFS and DFS.

In the remainder of this lecture, we will look at two examples of two graph searches applied on different graph classes. Section 2 focuses on Lexicographic Breadth First Search and Chordal graphs. Section 3 focuses on Lexicographic Depth First Search and Cocomparability graphs.

2 Lexicographic Breadth First Search & Chordal Graphs

2.1 Chordal Graphs

Definition 3. *A graph $G(V, E)$ is chordal if the largest induced cycle in G is a triangle.*

Chordal graphs are sometimes referred to as *triangulated graphs* as well. Convince yourself that chordality is a hereditary property. This means if G is chordal, so is every induced subgraph of G.

Theorem 3. *Chordal graphs are perfect.*

Proof. By the Strong Perfect Graph Theorem, it suffices to show that if G has no hole or antihole. It is easy to see that G has no holes since the largest cycle is a triangle. Suppose G contains an antihole $\overline{C}$. Let C be the complement of $\overline{C}$ in $\overline{G}$. C is an odd cycle with 5 or more vertices. Notice first that the complement of a C_5 is a C_5 and thus $\overline{C} \neq C_5$ since G is chordal. Any other odd cycle $C_{k\geq 7}$ in $\overline{G}$ must have two edges ab, cd where a and b are both not adjacent to c and d. In $\overline{C}$, $abcd$ forms a C_4, thereby contradicting G being chordal. □

Theorem 4. *Every minimal separator of a chordal graph is a clique.*

Proof. Let $G(V, E)$ be a chordal graph. If G is complete, then claim clearly holds. Suppose G is not complete. Let S be a minimal separator of G. Suppose S is not a clique. This means there exists two vertices $u, v \in S$ such that $uv \notin E$. Since S is a separator, $G \backslash S$ has two or more connected components. Let C_1, C_2 be two of these connected components. Since $u \in S$, there must exist two vertices $a \in C_1, b \in C_2$ such that u belongs to an a, b path, for otherwise $S \backslash \{u\}$ is a smaller separator than S (a contradiction to the minimality of S). Similarly, there must exist two vertices $c \in C_1, d \in C_2$ such that v belongs to a c, d path. Since $a, c \in C_1$, let P_1 be an induced a, c path in C_1 (convince yourself such a path must exist), and let P_2 be an induced d, b path in C_2. Consider the subgraph induced by P_1, P_2, u and v. This subgraph is a cycle of length at least 4 (when $a = c, b = d$, we have $C_4 = a, u, b, v$). A contradiction again to G being chordal. Therefore $uv \in E$ and S is a clique. □

Theorem 5. *If G is chordal, then either G is complete or G has at least two non-adjacent simplicial vertices.*

Proof. The proof is by induction on the size of the graph, i.e. the number of vertices. If $|V| = 1$, then G is clearly complete. Suppose $|V| > 1$ and G is not complete, then G has a minimal separator S. By Thereom 4, S is a clique.

Let C_1, C_2 be two connected components of $G \backslash S$. Consider the subgraph $G_1 = C_1 \cup S$ of G, since chordality is a hereditary property, G_1 is chordal and by induction hypothesis, G_1 is either complete or has at least two non-adjacent simplicial vertices. Either way G_1 has a simplicial vertex that remains simplicial in G (why?). Similarly $G_2 = C_2 \cup S$ is chordal and has at least one simplicial vertex that remains simplicial in G. Therefore G has two non-adjacent simplicial vertices. □

Corollary 1. *G is chordal iff every induced subgraph of G has a simplicial vertex.*

Proof. left as an exercise.

□

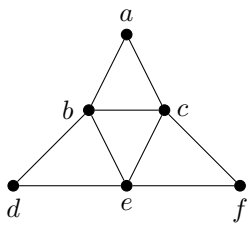
2.2 Perfect Elimination Orders

Given a graph $G(V, E)$, and a total ordering $\sigma = v_1, \ldots, v_n$ of V. Let $N^-(v_i)$ denote the neighbours of v_i that appear to the left of v_i in σ. Define $N^+(v_i)$ analogously.

$$N^-(v_i) = \{v_j : v_j v_i \in E \text{ and } j < i\}$$
$$N^+(v_i) = \{v_j : v_j v_i \in E \text{ and } i < j\}$$

For two vertices v_i, v_j such that $i < j$, we write $v_i \prec_\sigma v_j$ to denote that v_i is the left of v_j in σ. We drop the subscript if σ is clear in the context, and write $v_i \prec v_j$.

Definition 4. *A vertex ordering $\sigma = v_1, v_2, \ldots, v_n$ is a **perfect elimination order** - or PEO - if for all $i \in [n], v_i$ is simplicial to the left in σ, i.e. $N^-(v_i)$ is a clique.*



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

Figure 1: Example of a PEO ordering on a graph G

Surprisingly, one can use PEO to characterize chordal graphs.

Theorem 6. *G is chordal if and only if G has a perfect elimination order.*

Proof. Suppose G has a PEO σ but is not chordal. Let $\mathcal{C}$ be an induced cycle in G on 4 or more vertices. Let w be the vertex of $\mathcal{C}$ that appears last in σ. Then w must have at least two predecessors in σ (namely its neighbours in $\mathcal{C}$), call them u, v. Then $u, v \prec_\sigma w$ and $uv \notin E$. A contradiction to w being simplicial to the left in σ.

Conversely, let G be a chordal graph. By Theorem 5, G has a simplicial vertex. Call it v_n. Since chordality is a hereditary property, $G - v_n$ is also chordal. An induction on the size of G now concludes the proof. In particular, let $\sigma' = v_1, v_2, \ldots, v_{n-1}$ be a PEO of $G - v_n$, then $\sigma = \sigma' \cdot v_n$[1] is a PEO of G since v_n is simplicial in G. □

Applications: One of the main problems in structural graph theory is *graph recognition*: Given a class of graphs $\mathcal{G}$ that satisfies some sort of structure, and a graph G, is it easy/hard to check if $G \in \mathcal{G}$? Since PEOs fully characterize chordal graphs (Theorem 6), chordal graph recognition reduces to computing a PEO (or at least one way to recognize this graph family would be to compute a PEO). Before looking at the complexity of computing such orderings, let's see what else we can use them for.

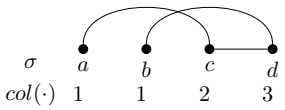
Consider the following algorithm:

Algorithm 1 GreedyCOL

Input: An arbitrary graph $G(V, E)$ and an ordering σ of V.
Output: A proper colouring of the vertices of G.

```
for i = 1...n do
    assign v_i the smallest colour not used among N⁻(v_i).
end for
```

Running *GreedyCOL* on the ordering below produces a 3-colouring where a 2-colouring exists. It's in fact easy to construct graphs where this algorithm behaves arbitrarily bad. Surprisingly, if *GreedyCOL* is given a "good" ordering, it produces an optimal colouring for certain graph families, chordal being one of them, as shown in Theorem 7 below.



Theorem 7. *If σ is a PEO, algorithm* GreedyCOL *gives an optimal colouring.*

Proof. Consider a vertex v_i in σ. Since v_i has $|N^-(v_i)|$ left neighbours, at least one of the colours $1, \ldots, |N^-(v_i)|+1$ is not used among $N^-(v_i)$. Therefore the maximum number of colours used by GreedyCOL is $\max_i |N^-(v_i)|+1$.

Let v_i^* be the vertex that achieves $\max_i |N^-(v_i)| + 1 = |N^-(v_i^*)| + 1$. We therefore have

$$\chi(G) \leq |N^-(v_i^*)| + 1 \tag{1}$$

Since σ is a PEO, v_i^* is simplicial to its left in σ and so $N^-(v_i^*)$ forms a clique, and thus

$$\omega(G) \geq |N^-(v_i^*)| + 1 \tag{2}$$

We know that

$$\omega(G) \leq \chi(G) \tag{3}$$

Combining (1), (2), and (3) we get

$$\chi(G) \leq |N^-(v_i^*)| + 1 \leq \omega(G) \leq \chi(G)$$
$$\chi(G) = \omega(G) = |N^-(v_i^*)| + 1$$

[1] The concatenation of σ' and v_n

Therefore algorithm *GreedyCOL* produces an optimal colouring on G. □

Corollary 2. *Algorithm* GreedyCOL *can be tweaked to compute a maximum clique on G if σ is a PEO.*

Proof. The proof follows from perfection. □

Exercise: Prove that the following algorithm computes a maximum independent set if G is chordal and σ a PEO.

Algorithm 2 GreedyIS

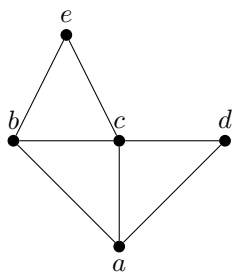
Input: An arbitrary graph $G(V, E)$ and an ordering σ of V.
Output: An independent set S

1: $S = \emptyset$
2: **for** $i = n \dots 1$ **do** ▷ Notice we're scanning σ in reverse order.
3: Add v_i to S if none of its neighbours appear in S already.
4: **end for**

The question remains then: How do we come up with a PEO? Is it NP-hard to compute such an ordering? **No!** In fact there is a simple elegant algorithm to compute such an ordering in *drum roll* linear time *drum roll* !

2.3 Lexicographic Breadth First Search

Lexicographic breadth first search (LexBFS) is a variant of BFS, that assigns lexicographic labels to vertices, and uses said labels to break any ties that might occur. Consider the graph below for instance.



$\sigma = a, d, b, c, e$ σ is a BFS, vertices b, c, d tied.
$\pi = a, d, c, b, e$ π is a LexBFS, vertices c and d are not tied any more.

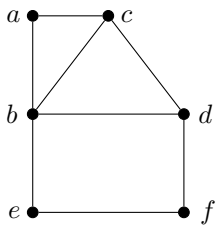
Figure 2: BFS vs. LexBFS

LexBFS would modify BFS to visit vertices with "stronger previous pull" first. In the example above for instance, once a, d were visited, we have $N^-(b) \subset N^-(c)$, and thus we're forced to visit c before b because vertex c is "pulled" by d. This condition alone - on the size of left neighbourhoods - does not actually characterize LexBFS. In addition to left neighbourhoods, LexBFS takes into account the ordering of the vertices in σ. In particular, every LexBFS is first a BFS, and thus must respect the "breadth first" condition before checking the size of left neighbourhoods. Algorithm 3 below formally describes this graph search.

Algorithm 3 LexBFS

Input: A graph $G(V, E)$ and a start vertex s
Output: An ordering σ of V

```
assign the label ϵ to all vertices, and label(s) ← {n + 1}
for i ← 1 to n do
    pick an unnumbered vertex v with lexicographically largest label
    σ(i) ← v                                   ▷ v is assigned the number i
    foreach unnumbered vertex w adjacent to v do
        append(n − i) to label(w)
    end for
end for
```



$\sigma(i)$	Affected Vertices	σ
$\sigma(1) = d$	$label(b) = label(c) = label(f) = 5$	d
$\sigma(2) = c$	$label(b) = 54$ and $label(a) = 4$	d, c
$\sigma(3) = b$	$label(a) = 43$ and $label(e) = 3$	d, c, b
$\sigma(4) = f$	$label(e) = 32$	d, c, b, f
$\sigma(5) = a$		d, c, b, f, a
$\sigma(6) = e$		d, c, b, f, a, e

Figure 3: A step by step computation of a LexBFS ordering on G starting at vertex d.

LexBFS was introduced by Rose, Tarjan and Lueker [6] to recognize chordal graphs. In particular they proved the following theorem:

Theorem 8. *Let $G(V, E)$ be a chordal grpah, then LexbFS(G) is a PEO.*

In order to prove Theorem 8, we use the following characterization of LexBFS orderings given by Dragan, Falk and Brandstädt [3]:

Theorem 9. *[The LexBFS 4 Point Condition] Let $G(V, E)$ be an arbitrary graph, and σ an ordering of V. σ is a LexBFS ordering if and only if for every triple $a \prec b \prec c$, if $ac \in E, ab \notin E$, then there exists a vertex d such that $d \prec a$ and $db \in E, dc \notin E$.*

The triple abc as defined above is called a bad (needy? :) LexBFS triple, and vertex d a private neighbour of b with respect to c. It is easy to see that such a vertex d must exist and be a neighbour of b since σ is a BFS. The fact that $dc \notin E$, ensures that b must indeed be pulled first before c, despite the pull of a on c. Intuitively, this just captures the lexicographic variant.

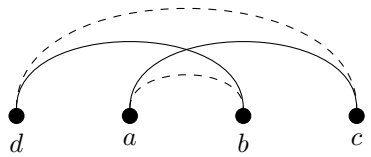


Figure 4: The LexBFS 4 Point Condition

Using the LexBFS 4 Point Condition, it is now easy to prove Theorem 8.

Proof of Theorem 8. Let G be a chordal graph, and $\sigma = \text{LexBFS}(G)$. Suppose σ is a not a PEO. Choose a vertex x to be the left most (the first) vertex in σ where $N^-(x)$ is not simplicial. In particular, let $y, z \prec x$ be two neighbours of x such that $yz \notin E$. Without loss of generality, suppose $z \prec y$. By Theorem 9, there must exist a vertex w such that $w \prec z$ and $wy \in E, wx \notin E$. If $wz \in E$, the quadruple $wzyx$ forms a C_4, a contradiction to G being chordal. Therefore $wz \notin E$, in which case $w \prec z \prec y$ forms a needy/bad LexBFS triple. Again by Theorem 9, there must exist a private neighbour q of z with respect to y, and $q \prec w$. Using the chordality of G, it is easy to see that $qw \notin E$, thus creating yet another bad triple. We repeat the same argument over and over again, thereby contradicting the finiteness of G. Therefore x is simplicial, and σ a PEO. □

In the remainder of this section, we briefly discuss one way to implement LexBFS in linear time.

2.4 Partition Refinement

Definition 5. *Let $S = \{u_1, u_2, \ldots, u_k\}$ be a set of elements. A collection $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ is a **partition** of S if for all $i \neq j \in [k]$, $P_i \cap P_j = \emptyset$ and $\bigcup_{i=1}^{k} P_i = S$. The P_is are called **partition classes**.*

Definition 6. *Given a set S, a partition $\mathcal{P} = \{P_1, \ldots, P_k\}$ of S, and a subset $T \subseteq S$, we say that T **refines** $\mathcal{P}$ if $\forall i \in [k], P_i$ is replaced with sub-partitions A_i, B_i in this order, where $A_i = P_i \cap T, B_i = P_i \backslash T$.*

*This is known as **partition refinement**, where T is used to refine the partitions of S.*

One elegant and efficient way to implement LexBFS in linear time for arbitrary graphs (linear in the size of the graph, so $O(m+n)$ time where $m = |E|, n = |V|$) is by means of partition refinement [4]. The set S is V, and T is the neighbourhood of a vertex p, called a **pivot**. The algorithm goes as follows: Initially σ is empty. We start by letting $\mathcal{P} = \{P_1 = V\}$. We choose an arbitrary start vertex $s \in V$ as a pivot, and use $N(s)$ to refine $\mathcal{P}$. We append s to σ. Initially, we replace V with $A_1 = V \cap N(s), B_1 = V \backslash N(s)$. The new partition now looks like $\mathcal{P} = \{A_1, B_1\}$. Every time a pivot p is selected to refine, it is appended to the ordering.

In general, given a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, and a pivot p, we use $N(p)$ to refine $\mathcal{P}$ by creating the following new partition $\{P_1 \cap N(p), P_1 \backslash N(p), P_2 \cap N(p), P_2 \backslash N(p), \ldots, P_i \cap N(p), P_i \backslash N(p), \ldots, P_k \cap N(p), P_k \backslash N(p)\}$, while maintaining the order of the partition classes, and append p to σ. A vertex is eligible to be a pivot if and only if it belongs to the first partition class. The algorithm stops when all the partition classes are either empty or contain a single vertex.

Example: Consider the graph in Figure 3, we will use partition refinement to produce the same LexBFS ordering in the table of the same figure.

Initially we start with one partition class, $V = \{a, b, c, d, e, f\}$. We choose an arbitrary start vertex, d and refine as follows:

$$P_1 = V \cap N(d) = \{b, c, f\}$$
$$P_2 = V \backslash N(d) = \{a, e\}$$

$\sigma = d$ so far. Vertices in P_1 are all eligible to be pivots, we choose vertex c as the next pivot and refine

$\mathcal{P} = \{P_1, P_2\}$ as follows:

$$\begin{aligned} P_1 \cap N(c) &= \{b\} \\ P_1 \backslash N(c) &= \{f\} \\ P_2 \cap N(c) &= \{a\} \\ P_2 \backslash N(c) &= \{e\} \end{aligned}$$

$\sigma = d, c$. The new partition now is $\mathcal{P} = \{\{b\}, \{f\}, \{a\}, \{e\}\}$ in this order. Since all the partition classes are singletons, the refinement is done, and σ is d, c, b, f, a, e, the same LexBFS ordering produced in Figure 3. Try to convince yourself (prove) that every ordering produced by this refinement is indeed a LexBFS ordering of an (arbitrary) graph, in particular notice the parallelism of maintaining the ordering of the partition classes and appending lexicographic labels.

3 Lexicographic Depth First Search & Cocomparability Graphs

Next, we'll look at another example of a different graph search used on a different graph class.

3.1 Cocomparability Graphs

Cocomparability graphs are a large, well studied, graph family. It strictly contains a number of graph classes including interval graphs and permutation graphs. In fact, a classical characterization of interval graph is the following:

Theorem 10. *A graph $G(V, E)$ is an interval graph if and only if G is chordal and cocomparability.*

Cocomparability graphs are the complement of comparability graphs. A graph $G(V, E)$ is comparability graph if its edge set admits a transitive orientation. That is, there is a way to orient (single direction) the edges in E, such that for every triple a, b, c oriented $a \to b, b \to c$, there must exist an edge oriented $a \to c$. Cocomparability graphs are perfect, and thus by the Weak Perfect Graph Theorem, so are comparability graphs. Figure 5 below gives an example of a comparability and a non-comparability graph. Convince yourself that the graph on the left is indeed a comparability graph, by coming up with a transitive orientation of the its edges; whereas the graph on the right is not.

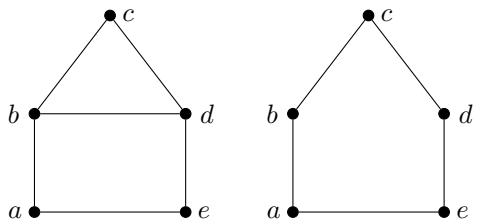


Figure 5: A comparability graph (to the left) and a non-comparability graph (to the right).

Cocomparability graphs can be characterized by cocomparability orderings, also known as *umbrella-free orderings*. In particular, a graph $G(V, E)$ is a cocomparability graph if and only if there exists an ordering σ of V such that for every triple $a \prec_\sigma b \prec_\sigma c$, if $ac \in E$ then either $ab \in E$ or $bc \in E$ or both. It is easy to see that umbrella free orderings are precisely transitive orientations of the comparability graph.

Side note: There is a close relationship between cocomparability/comparability graphs and partially ordered sets. A *partially ordered set*, or poset, $P(V, \prec)$ is an irreflexive, antisymmetric and transitive relation on the set V. In particular, two elements $a, b \in V$ are comparable if $a \prec b$ or $b \prec a$, otherwise they are incomparable, we write $a \| b$. because of transitivity, if three elements are comparable $a \prec b, b \prec c$ then $a \prec c$. This is precisely what a transitive orientation is on a comparability graph. In fact, if $G(V, E)$ is a comparability graph, then G together with a transitive orientation of E can equivalently be represented by a poset $P(V, \prec)$ where $ab \in E$ if and only if a and b are comparable in P. And thus umbrella-free orderings can too be represented by posets. This equivalence gives another way to solve problems for these graph families. For more on posets and order theory in general, check Tom Trotter's course [lecture 13 and beyond]. The Wiki page is a good start too.

3.2 Lexicographic Depth First Search

One can extend DFS to a lexicographic version as well, known as *lexicographic depth first search*. LexDFS (for short) was introduced in [2], and has since led to a number of efficient algorithms, especially on cocomparability graphs. We begin by looking at properties of this graph search, then give an example of its use; namely a certifying algorithm to compute a maximum independent set (MIS) on cocomparability graphs.

Formally, LexDFS assigns labels to vertices as they are being processed, ties are broken using the labels, where vertices with the highest label are chosen first. A similar idea to what LexBFS does, but as we will see, the labeling takes into account the "depth" aspect of the search as well. Algorithm 4 below is a formal description of this process.

Algorithm 4 LexDFS

Input: A graph $G(V, E)$ and a start vertex s
Output: An ordering σ of V

```
1: assign the label ϵ to all vertices, and label(s) ← {0}
2: for i ← 1 to n do
3:     pick an unnumbered vertex v with lexicographically largest label
4:     σ(i) ← v                                   ▷ v is assigned the number i
5:     foreach unnumbered vertex w adjacent to v do
6:         prepend i to label(w)
7:     end for
8: end for
```

Running the algorithm on the graph in Figure 2, gives the following ordering

$$\text{LexDFS}(G) = \sigma = a, b, c, e, d. \tag{4}$$

Notice in particular how we are forced to visit vertex e before vertex d. LexDFS can too be characterized by a 4 Point Condition, given by [2], which says:

Theorem 11. *[The LexDFS 4 Point Condition] Let $G(V, E)$ be an arbitrary graph, and σ an ordering of V. σ is a LexDFS ordering if and only if for every triple $a \prec b \prec c$, if $ac \in E, ab \notin E$, then there exists a vertex d such that $a \prec d \prec b$ and $db \in E, dc \notin E$.*

Vertex d is a private neighbour of b with respect to c. Intuitively, the theorem shows that despite vertex c having a pull from vertex a that b does not have, since σ is a LexDFS, there must exist a vertex later (deeper?) in the ordering that pulled b first. This vertex is d. The formal proof of the statement of the theorem is left as an exercise.

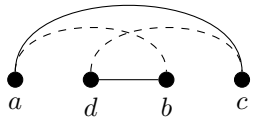


Figure 6: The LexDFS 4 Point Condition

Unfortunately LexDFS cannot be implemented in linear time - yet - for arbitrary graphs; partition refinement is indeed one way to do so, but it requires the reshuffling/sorting of the partitions, and this sorting is the bottleneck to a linear time algorithm. However, linearity is achieved for specific graph families, cocomparability being one of them [5].

Multi-Sweep Algorithms: For the purpose of this (MIS) problem, we need to introduce what we call multi-sweep algorithms. These are algorithms that compute a sequence of orderings[2] $\sigma_1, \sigma_2, \ldots$ where σ_i is used to compute σ_{i+1}. This means, that if there are additional ties between vertices when computing σ_{i+1}, the algorithm uses σ_i to break them using some specific rule. We will focus on the so called $^+$ rule; where ties are broken by choosing the right most vertex in σ_i. Formally, we write $\tau = \text{LexDFS}^+(G, \sigma)$, where σ is some ordering of G, and τ is a LexDFS ordering of G that uses σ to break ties by always choosing the right most vertex in σ. Consider for instance, the graph in Figure 2. Let $\sigma = a, b, c, e, d$ be the ordering computed earlier in (4). A $\text{LexDFS}^+(G, \sigma)$ of G is the unique ordering $\tau = d, c, a, b, e$: vertex d was chosen first because it was the right most vertex in σ, c was second because it was the right most eligible vertex in σ, etc.

One nice property of $^+$ sweeps is the preservation of umbrella-free orderings; this means:

Theorem 12. *Let $G(V, E)$ be a cocomparability graph, and σ an arbitrary umbrella-free ordering. The ordering $\tau = \text{LexDFS}^+(G, \sigma)$ is an umbrella-free ordering. We call τ a LexDFS umbrella-free ordering.*

For a proof of this result, see [1]. In fact, the authors characterize all the graph searches that preserve umbrella-free orderings.

Combining Theorems 11 and 12, one can deduce the LexDFS C_4 property of cocomparability graphs (Figure 7). Let τ be an umbrella-free ordering. Let abc be a bad triple in τ. This means $a \prec_\tau b \prec_\tau c$, where $ac \in E, ab \notin E$. Since τ is an umbrella-free ordering, it follows that $bc \in E$. Since abc is a bad LexDFS triple, there must exist a private neighbour d of b with respect to c, such that $a \prec_\tau d \prec_\tau b$ and $db \in E, dc \notin E$. The edge $ad \in E$ is now forced, for otherwise, we would have $a \prec_\tau d \prec_\tau c$ and $ac \in E$ and both $ad, dc \notin E$. A contradiction to τ being umbrella-free.

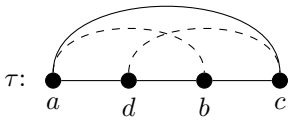


Figure 7: The LexDFS C_4 Property of cocomparability graphs.

3.3 Maximum Independent Set on Cocomparability Graphs

We now present a certifying algorithm to compute a maximum independent set (MIS) on cocomparability graphs. A certifying algorithm is an algorithm that, along with a solution, produces a certificate to check if said solution is indeed optimal. First, let's come up with a natural certificate for this problem.

[2]Either LexBFS, LexDFS or any other type of orderings

We note that computing a maximum independent set on cocomparability graphs is equivalent to computing a maximum clique in the corresponding comparability graph. Since both graph families are perfect, the clique number is the chromatic number in the comparability graph. So one way to certify maximality of the clique is to give a proper colouring that matches the size of the clique.

Let's look at the following example. Let $G(V, E)$ be a comparability graph, and $H(V, E') = \overline{G}$ the complement of G.

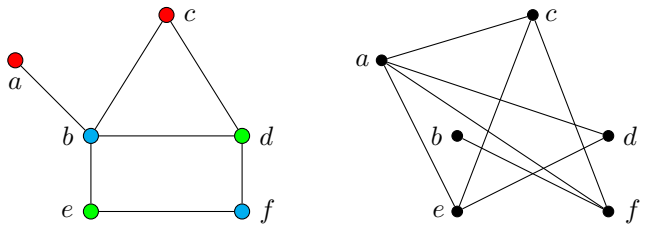


Figure 8: G a comparability graph on the left, and H the complement of G on the right

The graph in Figure 8 is a comparability graph, as witnessed by the following transitive order $\tau = b, a, f, c, e, d$. Since the largest clique in G is of size 3, namely $\{b, c, d\}$, it follows that the chromatic number of G, $\chi(G) = 3$, and a proper colouring is given by the coloured vertices above.

Every colour class (the set of vertices that received the same colour) forms an independent set in G. Therefore the colour classes of G form cliques in H. In fact, they form a clique cover of H. Since $\chi(G)$ is minimum, the number of colour classes in minimum and thus the clique cover in H is minimum. We thus have:

$$\chi(G) = \omega(G)$$
$$\alpha(H) = \kappa(H)$$

The clique cover number for any graph is always an upper bound on the independent set - since one can collect at most one vertex from each clique to place in the independent set, and these collected vertices are not necessarily pairwise independent. Therefore one way to certify the optimality of the MIS algorithm is to produce an independent set and a clique cover of equal size. We can do this because cocomparability graphs are perfect. This is precisely what we will do.

Algorithm 5 Maximum Independent Set in Cocomparability Graphs

Input: A cocomparability graph $G(V, E)$ and an umbrella-free ordering σ.
Output: An independent set S of G.
1: $\tau = \text{LexDFS}^+(G, \sigma)$
2: Scan τ right to left and greedily add vertices to S that do not already have neighbours in S.

Recall that a greedy collection of an independent set is the process of scanning the ordering -in this case right to left- and placing a vertex in S as long as none of its neighbours are already in S. We illustrate Algorithm 5 on the graph H in Figure 8 (redrawn differently) below (Figure 9). Notice (or check for yourself) that σ is indeed an umbrella-free ordering, but not a LexDFS ordering (doesn't need to be) - in particular $b \prec_\sigma a \prec_\sigma f$ is a bad LexDFS triple. The algorithm begins by placing b (right most in τ) in S, this "removes" f and a, the right most available vertex in τ is c. Thus $c \in S$, this removes e, then the right most eligible vertex is d. Thus $S = \{b, c, d\}$.

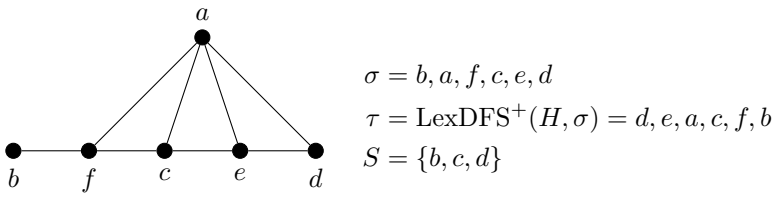


Figure 9: An illustration of the algorithm.

Theorem 13. *Algorithm 5 produces a maximum independent set of G.*

Proof. Let $\tau = u_1, u_2, \ldots, u_n$, and let $S = v_1, v_2, \ldots, v_k$ be the vertices in the independent set where $v_k \prec_\tau v_{k-1} \prec_\tau \ldots \prec_\tau v_2 \prec_\tau v_1 = u_n$. For every $i \in [k]$, let T_i be the set of vertices in τ from v_i (included) up to but not including v_{i+1}; $T_i = \{u_j | v_{i+1} \prec_\tau u_j \prec_\tau v_i\} \cup \{v_i\}$.

Clearly S is an independent set. Instead of proving the optimality of the S, i.e. that it is of maximum size, we'll produce a clique cover of equal cardinality. In particular, we claim that for any $v_i \in S$, T_i forms a clique. If this is true, then each v_i belongs to a clique in G, the T_i's form a clique cover, and thus $|S| = \kappa(G)$. Thereby completing the proof.

Suppose there exists a vertex v_i such T_i is not a clique. Let $a, b \in T_i$ be two vertices such that $a \prec_\tau b \prec_\tau v_i$ and $ab \notin E$. Notice that v_i is different than both a and b, otherwise we contradict the choice of v_{i+1}.

This triple abv_i forms a bad LexDFS triple in τ where $ab \notin E, av_i, bv_i \in E$. By the LexDFS 4 Point Condition, there must exist a vertex d such that $a \prec_\tau d \prec_\tau b$ and $db \in E, dv_i \notin E$. However $a \prec_\tau d$ and $dv_i \notin E$ contradicts the construction of T_i and thus the choice of v_{i+1}. Therefore $ab \in E$ and T_i is a clique. □

An algorithm from the book. A proof from the book. :)

To go back to the example in Figure 9, the clique cover we get is $\{b, f\}, \{c, a, e\}, \{d\}$.

For the curious mind: Multi-sweep algorithms have led to a number of elegant results in algorithmic and structural graph theory. Below is a list of some other problems solved using graph searching on various graph classes - email me for references.

- Graph recognition for a number of graph families.
- Colouring, independent set, clique, clique cover.
- Longest path, Hamilton path, and minimum path cover (this latter is an generalization of the Hamilton path problem: What is the minimum number (k) of paths necessary to cover all vertices in G? For $k = 1$, we have a Hamilton path.)
- "Weighted Hamilton path", i.e. TSP path version, on proper interval graphs.
- Maximum matching in linear time.
- Domination - various type of domination problems. One example is the dominating pair problem: Given a graph $G(V, E)$, does there exist a pair $u, v \in V$, such that every $uv-$path *dominates* G? A path P dominates a graph, if every vertex in G is either on P or has a neighbour on P.
- Minimal/minimum triangulations: Given an arbitrary graph $G(V, E)$, what is the minimum number of edges one can add to turn G into a chordal graph?

References

[1] Derek G Corneil, Jérémie Dusart, Michel Habib, and Ekkehard Kohler. On the power of graph searching for cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 30(1):569–591, 2016.

[2] Derek G Corneil and Richard M Krueger. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008.

[3] Feodor F Dragan, Falk Nicolai, and Andreas Brandstädt. Lexbfs-orderings and powers of graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 166–180. Springer, 1996.

[4] Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1):59–84, 2000.

[5] Ekkehard Köhler and Lalla Mouatadid. Linear time lexdfs on cocomparability graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 319–330. Springer, 2014.

[6] Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.