CS 163 & CS 265: Graph Algorithms

Week 7: Perfection and width

Lecture 7a: Chordal graphs, perfect graphs, and lexicographic breadth-first search

David Eppstein
University of California, Irvine

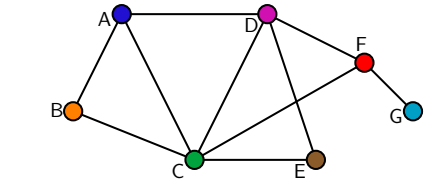
Winter Quarter, 2024

Chordal graphs

Last time...

Defined interval graphs, representing the intersections of intervals

Greedy coloring algorithm gives them first available color,
when the intervals are ordered by their left endpoints



Key property for coloring: Earlier neighbors form a clique (all adjacent), so when we use color k, we need $\geq k$ colors

Can we find this ordering just from the graph, without already knowing the intervals of an interval representation?

Chordal graphs

A graph is a chordal graph if and only if

- Every cycle longer than three vertices has a chord, another edge connecting two vertices of the cycle

  or

- It has an elimination ordering: an ordering of the vertices where the later neighbors of every vertex form a clique

  (Reverse this to get the ordering we need for greedy coloring)

  or

- For every two vertices x and y and every minimal subset S whose deletion would separate x from y, S is a clique

  or

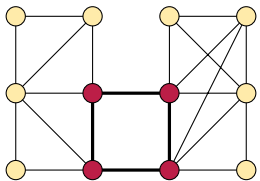
- It represents the intersections of a family of subtrees of a tree

  (We won't prove this one)

Elimination orderings $\Rightarrow$ no holes

A hole is a cycle of length ≥ 4 that does not have a chord (another edge connecting two vertices of the cycle)

So the first definition of chordal graphs is "graphs with no holes"



If C is the set of vertices in a hole, none of the vertices in C can be the first one picked in an elimination ordering

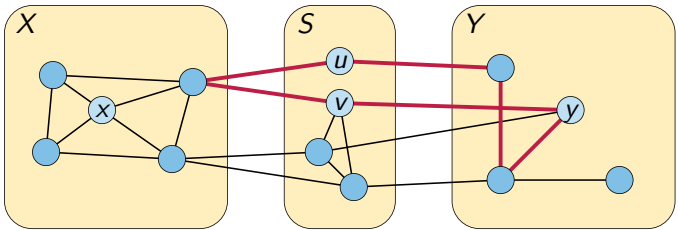
Whichever vertex you choose first has two later neighbors in C that are not adjacent to each other, so its later neighbors cannot form a clique

No holes $\Rightarrow$ clique separators

Let x, y be any two non-adjacent vertices in non-complete graph G

Let S be minimal separating them (no subset of S separates)

Suppose S is not a clique: some two vertices u, v not adjacent



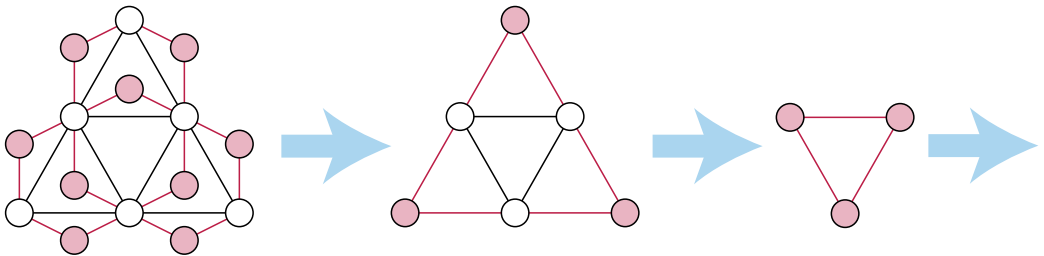
Both u and v have neighbors in X and Y, else could be removed

Shortest path from u to v through X +
shortest path from u to v through Y forms a hole

Easy (but slow) elimination ordering algorithm

While the graph is non-empty:

- Find a vertex v whose neighbors form a clique
- Output v and delete it from the graph



Cannot make a mistake, choose the wrong vertex, and get stuck:

If G has an elimination ordering, so does $G - v$
(the same ordering with v removed from it)

Clique separators $\Rightarrow$ elimination orderings

If the whole graph has the property that all minimal separators are cliques, so do the subgraphs formed by deleting vertices during the elimination ordering process

If this subgraph is a clique, we can choose any vertex next

Otherwise, we can find a vertex whose neighbors form a clique by finding a minimal separator of any two non-adjacent vertices and recursing on one side of the separator

So the easy elimination order algorithm can never get stuck!

Linear-time elimination ordering

Basic idea of linear-time algorithm

Find a special breadth-first search ordering *

Reverse it

Prove that, in chordal graphs, it will be an elimination ordering

Check that it really is an elimination ordering

[Rose et al. 1976]

* the ordering could be produced by breadth-first search,
but we will use a totally different algorithm to find it

How to tell whether it is an elimination order?

Our ordering algorithm will produce an ordering of any graph,
but it will only be an elimination order if the graph is chordal

(just like reverse postorder DFS produces an ordering of any graph but is only a topological order if the graph is a DAG)

Slow test for being an elimination order: for each v, and each two later neighbors x and y, check that x and y are adjacent

Faster (linear time):

for each vertex v:
    let w be the neighbor of v that comes next in the order
    check that all other later neighbors of v are neighbors of w

(Other pairs x, y of later neighbors don't need to be checked by v; we can procrastinate checking until we reach w or even later)

How to tell if an ordering comes from BFS

Recall that BFS can list vertices in different orders depending on how it loops over the neighbors of each vertex

Given any ordering of the vertices of a connected graph, define parent(v) to be the first neighbor of v listed in the ordering, or v itself if it has no earlier neighbor

Then the ordering comes from BFS (for some ordering of adjacency lists of each vertex) if and only if the vertices are in sorted order according to the positions of their parents

- If two vertices have different parents, the one with the earlier parent must come first
- Two vertices with the same parent can occur in either order

So vertices are sorted by the positions of their parents

Lexicographic orderings

Given an ordering of vertices, define $\text{pred}_i(v)$ to be the ith neighbor of v, among all earlier neighbors, or v itself if there are fewer than i neighbors earlier than v

parent = pred_1 but we can use other values of pred_i to break ties:

- If two vertices have different parents (pred_1), the one with the earlier parent must come first (same as BFS)
- If they have the same pred_1 but different values of pred_2, the one with earlier pred_2 must come first
- If they have the same pred_1 and pred_2 but different values of pred_3, the one with earlier pred_3 must come first
- ...

So they are sorted by the lexicographic ordering on the tuple of positions of $\text{pred}_1, \text{pred}_2, \text{pred}_3, \ldots$

Algorithm for finding lexicographic order (LexBFS)

Maintain a sequence of sets of unprocessed vertices $S_1, S_2, \ldots$, initially all in one big set

While sequence is nonempty:

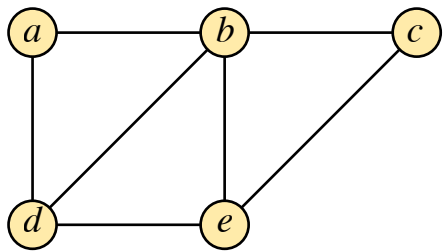
- Let S_i = first set in the sequence, v = any vertex in S_i
- Remove v from S_i (also remove S_i if it becomes empty)
- Output v; let $N(v)$ = set of neighbors of v
- For each set S_j in the sequence, split S_j into two sets $S_j \cap N(v)$ and $S_j \setminus N(v)$ (in that order in the sequence)

Return the vertices in the order given by the sequence

Example: Not all BFS orderings are lexicographic

In the graph below...

- BFS can give a, b, d, c, e
- Not a valid lexicographic order!
- $\text{pred}_1(e) = \text{pred}_1(c) = b$, but $\text{pred}_2(e) = d, \text{pred}_2(c) = c$
- $\therefore e$ must come before c
- LexBFS gives a, b, d, e, c



Data structures needed to make this work

With some care, we can maintain the following structures and handle each vertex v in time $O(|N(v)|)$ (giving linear time overall):

- Dynamic sets S_i of vertices (e.g. Python set)
- Dynamic sequence of sets (e.g. linked list, not Python list)
- Dictionary mapping each vertex to the set that contains it
- Dictionary mapping sets that are being split to the second new set being split from them

The same "partition refinement" data structures are also useful in finite automaton minimization, parallel scheduling, and layered graph drawing

Why does this give an elimination ordering?

The proof involves a lot of messy case analysis
(not very enlightening)

But it also proves a more general claim:

- Let G be any graph, not necessarily chordal
- Run the same algorithm: LexBFS, then reverse the order
- Add set X of "missing" edges preventing it from being an elimination order: if any vertex v has two later neighbors u and w that are not adjacent, (u, w) is missing
- Then $G + X$ is a minimal chordal completion of G: it's chordal and no subset of X has the same property

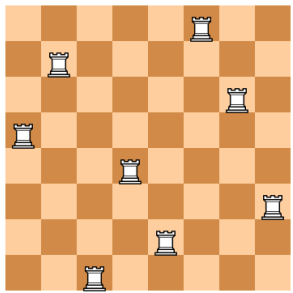
Perfect graphs

Perfect graphs

A graph is perfect if:

- Optimal # colors equals # vertices in largest clique
- Deleting any vertex produces another perfect graph (defined recursively the same way)

True of interval graphs, chordal graphs, and bipartite graphs (deleting any vertex produces another graph in the same class)



Also true for vertices = squares of chessboard, edges = rook moves

The strong perfect graph theorem

A graph G is perfect if and only if:

- It has no odd-length holes
- Its complement (the graph that has an edge wherever G does not) has no odd-length holes

Long conjectured [Berge 1961], finally proved by Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas (2006)

Implies the "weak perfect graph theorem" that complements of perfect graphs are perfect

If G is bipartite, coloring of complement = matching in G + single vertices, and clique in complement = independent set in G, so weak theorem implies Kőnig's theorem

Algorithms for perfect graphs

We can recognize perfect graphs in polynomial time ...

... but the polynomial is $O(n^9)$ [Chudnovsky et al. 2005]

We can optimally color and find cliques in perfect graphs in polynomial time ...

But the algorithm involves using the ellipsoid method for a linear program with exponentially many constraints [Grötschel et al. 1988]; no simple combinatorial algorithm is known

Morals of the story

Interval graphs $\subset$ chordal graphs $\subset$ perfect graphs

LexBFS algorithm for finding elimination orderings

The perfect graph theorem

Many open problems on algorithms for perfect graphs

References

Claude Berge. Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 10:114, 1961.

Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vušković. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005. doi:10.1007/s00493-005-0012-8.

Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1):51–229, 2006. doi:10.4007/annals.2006.164.51.

Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, 1988. doi:10.1007/978-3-642-97881-4.

Donald J. Rose, Robert E. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.