CS762: Graph-Theoretic Algorithms
Lecture 10: Recognizing interval graphs. Perfect graphs
January 28, 2002

Scribe: Hubert Chan

Abstract

In this lecture, we complete our discussion on recognizing interval graphs by introducing PQ-trees, and give a sketch of how it is used in recognizing interval graphs. We introduce the class of perfect graphs, and the perfect graph theorem.

1 Introduction

As seen in previous lectures, many NP-complete graph problems become easy when restricted to interval graphs. However, in order to take advantage of the algorithms for solving these problems, one must be able to determine if a given graph is an interval graph.

In this lecture, we introduce PQ-trees, defined by Booth and Lueker [BL76]. This is a data structure which stores permutations of a sequence of elements. It can be used to recognize interval graphs in linear time, and to create a set of intervals which generate the graph. We only give a sketch of the algorithm; the full details can be found in Booth and Lueker.

We also give a summary of other linear-time algorithms for recognizing interval graphs.

Finally, we define the class of perfect graphs, which is contains some of the graph classes that we have already seen, such as chordal and comparability graphs. We define the class of co-perfect graphs, and state the perfect graph theorem [Lov72], which implies that the classes of perfect and co-perfect graphs are equivalent.

2 PQ-trees

A *PQ-tree* is a data structure which efficiently stores a set of possible permutations by specifying a set of constraints on the permutations. The PQ-tree stores all permutations which satisfy the constraints.

A PQ-tree has two types of nodes: P-nodes, and Q-nodes, which are represented graphically as in figure 1. A Q-node specifies that its children can be placed in forward or reverse order, while a P-node specifies that its children can be placed in any order. (Note that if a node has two children, it does not matter if it is a P- or a Q-node.) The permutations stored by the entire tree, then, is the set of all allowed permutations of the leaves.

3 Recognizing interval graphs

Recall from the previous lecture that recognizing interval graphs can be reduced to determining if the set of all maximal cliques in a graph can be ordered such that for every vertex, the cliques

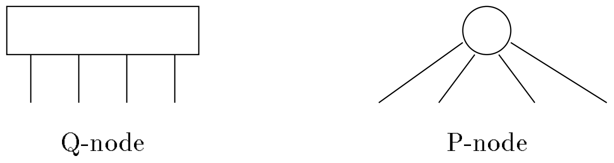

Figure 1: graphical representations of P- and Q-nodes

which contain it are consecutive. To solve this problem with a PQ-tree, let X, the leaves of the PQ-tree, be the set of all maximal cliques in a graph. Define $\mathcal{I}$, the consecutiveness constraints, as $\mathcal{I} = \{I_v | v \in V\}$, where I_v is the set of cliques which contain v. We give only a sketch of the algorithm. The full details are in Booth and Lueker [BL76].

We start with the PQ-tree allowing all possible permutations (i.e., it consists only of all the leaves connected to a P-node). We then add each constraint in $\mathcal{I}$ one at a time.

To add a constraint $I_v \in \mathcal{I}$, we modify the tree until the elements of I are consecutive. This can be done in $O(|I_v|)$ amortized time. If this cannot be done, then there is no possible ordering of the cliques which satisfies the constraints. If it can be done, we then update the tree to reflect the new constraint. Booth and Lueker showed that this can be done with only a constant number of replacement rules in $O(|I_v|)$ amortized time. Therefore, in total, adding a constraint can be done in $O(|I_v|)$ amortized time.

For example, let $X = \{A, B, C, D\}$, and $\mathcal{I} = \{I_1 = \{A, B, C\}, I_2 = \{A, D\}\}$. Figure 2 shows the steps in adding the constraints to the PQ-tree. The final tree gives all possible orderings which satisfy the constraints: $\{B, C, A, D\}$, $\{C, B, A, D\}$, $\{D, A, C, B\}$, and $\{D, A, B, C\}$. There are four possible orderings since the tree has two Q-nodes, and the elements of each Q-node can be in either forward or reverse order.

In all, determining whether or not a the set of maximal cliques has a "good" ordering can be done in time and space $O(|X| + \sum_{v \in V} |I_v|)$. However, chordal graphs have at most n maximal cliques, so $|X| \leq n$. Also, each vertex in a chordal graph can only be in no more maximal cliques than its degree (i.e. $|I_v| \leq \deg(v)$). Therefore $O(|X| + \sum_{v \in V} |I_v|) = O(n + m)$.

Note that after the algorithm is run, if it was successful in adding all constraints, the PQ-tree will store all allowed orderings for the maximal cliques. From this, as seen in the previous lecture, we can get a set of intervals which generates the graph.

4 Other algorithms for recognizing interval graphs

The algorithm presented in this lecture was the first linear-time algorithm for recognizing interval graphs. Korte and Möhring [KM89] later developed an algorithm based on modified PQ-trees. Hsu and Ma [HM91], and Hsu on his own [Hsu92] showed an algorithm which did not use PQ-trees.

Habib, Paul, and Vincent demonstrated an algorithm for recognizing interval graphs in an unpublished paper using a modified lexBFS, and Corneil, Olariu and Stewart [COS98] recognized interval graphs by running lexBFS five times, using different criteria for breaking ties, and starting each subsequent run at the last vertex chosen in the previous run.

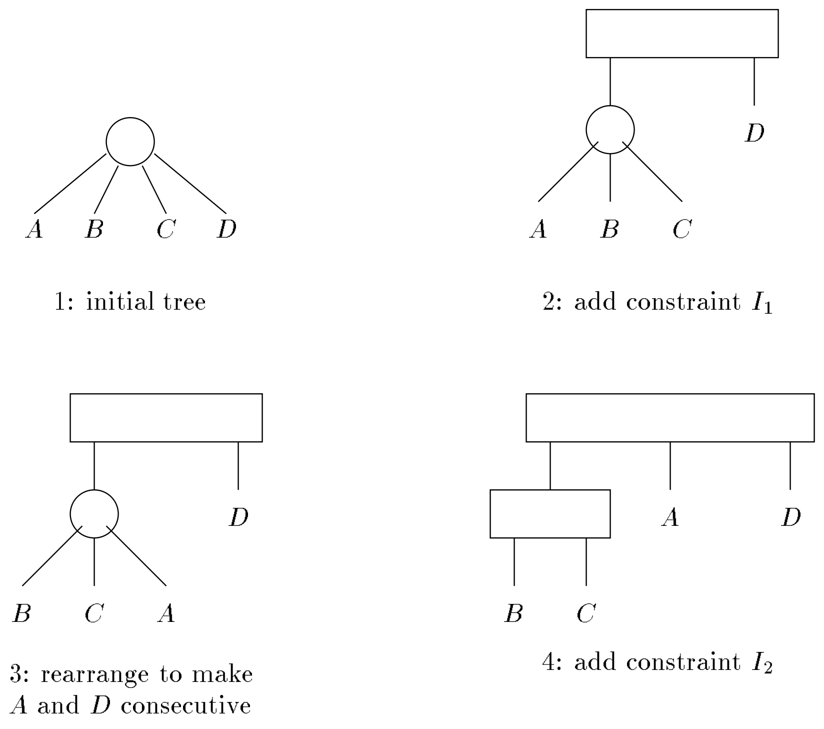

Figure 2: example of adding constraints to a PQ-tree

5 Perfect graphs

Definition 1 *A graph class $\mathcal{C}$ has the* hereditary property *if every induced subgraph of a graph in $\mathcal{C}$ is also in $\mathcal{C}$. For example, the classes of interval graphs, chordal graphs, and comparability graphs all have the the hereditary property.*

Definition 2 *A graph G is* perfect *if $\omega(G) = \chi(G)$, and $\omega(H) = \chi(H)$ for every induced subgraph H of G.*

Note that, by definition, the class of perfect graphs has the hereditary property. Also, if $\mathcal{C}$ is hereditary, and $\omega(G) = \chi(G)$ for every $G \in \mathcal{C}$, then the graphs in $\mathcal{C}$ are perfect. This means that interval graphs, chordal graphs, and comparability graphs are perfect.

Definition 3 *$\alpha(G)$, the* stability number *or* independence number, *is the size of the maximum independent set of G.*

Definition 4 *$\kappa(G)$, the* clique cover number, *is the smallest number of cliques in G such that each vertex of G is in a clique.*

Note that since each independent set can contain no more than one vertex from a given clique, $\alpha(G) \leq \kappa(G)$ for all graphs G.

Definition 5 *A graph G is* co-perfect *if $\alpha(G) = \kappa(G)$ and $\alpha(H) = \kappa(H)$ for every induced subgraph H of G.*

$$\begin{tikzcd}
 & \text{interval} \arrow[dl] \arrow[dr] & & & \\
\text{chordal} \arrow[dr] & & \text{co-comparability} \arrow[dl] & & \text{comparability} \arrow[dlll] \\
 & \text{perfect} & & &
\end{tikzcd}$$

Figure 3: Relationship between graph classes

Note that for every graph G, $\alpha(G) = \omega(\bar{G})$ (every independent set in G is also a clique in $\bar{G}$ and vice versa), and $\kappa(G) = \chi(\bar{G})$ (colour each vertex in a clique the same colour, and vertices in different cliques with different colours; so in the complement, no two vertices of the same colour will be adjacent). Therefore G is co-perfect if and only if $\bar{G}$ is perfect.

Theorem 1 *(Lovász [Lov72] — the perfect graph theorem) A graph is perfect if and only if its complement is perfect.*

This, along with the note above, implies that a graph is perfect if and only if it is co-perfect. It also implies that co-comparability graphs are perfect, since comparability graphs are perfect.

Figure 3 shows the relationship between the graph classes which we have seen so far: interval graphs, chordal graphs, (co-)comparability graphs, and perfect graphs. The arrows depict the subset relationship. Since the subset relationship is transitive, we omit the arrow from interval graphs to perfect graphs.

References

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using P-Q tree algorithms. *Journal of Computer and System Sciencs*, 13:335–379, 1976.

[COS98] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The ultimate interval graph recognition algorithm? In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-98)*, pages 175–180, New York, January 25–27 1998. ACM Press.

[HM91] Wen-Lian Hsu and Tze-Heng Ma. Substitution decompositions on chordal graphs and applications. In H.-L. Hsu and R. T. C. Lee, editors, *Lecture Notes in Computer Science*, volume 557, pages 52–60. 1991.

[Hsu92] Wen-Lian Hsu. A simple test for interval graphs. *Lecture Notes in Computer Science*, 657:11–16, 1992.

[KM89] Norbert Korte and Rolf H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, February 1989.

[Lov72] Lásló Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2:253–267, 1972.