CS762: Graph-Theoretic Algorithms
Lecture 1: Introduction
January 7, 2002

Scribe: Therese Biedl

Abstract

This lecture covered administrivia of the course (which will not be given here; see the web page for details), and then gave a general introduction to the topics of this course.

1 Introduction

This course is concerned with special graph classes, and problems that become easy/easier or still stay hard on these graphs classes.

To explain what we mean by that, we would have to review various notions, in particular graphs, graph algorithms, asymptotic analysis, and NP-hardness. This will *not* be done in this course; you are assumed to be familiar with them. If you want to review some material, the book by Cormen et al. [CLRS00] is a good source. Also, refer to [AGA99], a PostScript file containing part of the lecture notes of a previous offering of a similar course.

2 Some graph classes

In the following, we'll give an overview of graph classes that will be covered in this course. We will come back to all these graph classes in future lectures.

- *Interval graphs*: A graph is an interval graph if it can be represented as intersection graph of intervals. More precisely, given a set of intervals, we can define a graph by taking one vertex for every interval, and an edge between two vertices if and only if the two intervals intersect. A graph is an interval graph if it can be obtained this way. See Figure 1.

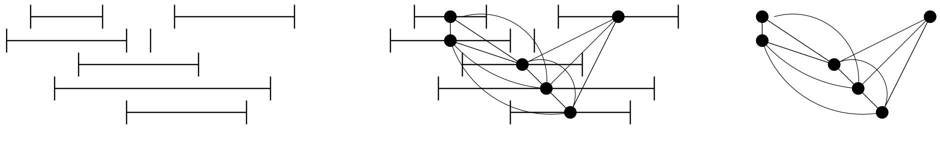

Figure 1: A set of intervals and the interval graph defined by it.

  We will also study various related graph classes, for example chordal graphs and perfect graphs. The main reference for this part of the course will be the book by Golumbic [Gol80].

- *Trees*: A graph is a tree if it is simple, connected, and has no cycle. See Figure 2.

  We will also study various related graph classes, in particular partial k-trees. The main reference for this part of the course will be the article by Bodlaender [Bod93].

- *Planar graphs*: A graph is called a planar graph if it can be drawn (in two-dimensional space) without a crossing. See Figure 2. Note that every tree is a planar graph, but the reverse is not true.

  We will also study various related graph classes, in particular outer-planar graphs and series-parallel graphs (SP-graphs). The material for this part of the course is assembled from many source, one possible reference is the book by Nishizeki and Chiba [NC88].

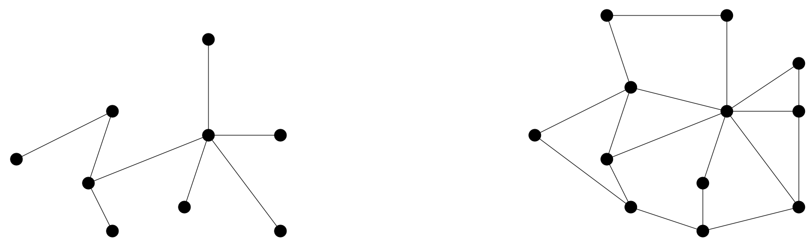

Figure 2: A tree and a planar graph that isn't a tree.

Unless explicitly said otherwise, all graphs that are studied in this class are *simple* (no multiple edges, no loops), *connected* (a path between any pair of vertices), and have at least 2 vertices (or whatever small number of vertices is needed to make the graph non-trivial).

Since the objective of this course is algorithms, these assumptions are justified. Most algorithms that work for simple graphs can be generalized to arbitrary graphs by first deleting all multiple edges and all loops.[1] Similarly, if a graph is not connected, then we can run the algorithm for all connected components. Finally, if the graph has just a few vertices, no advanced algorithm is needed.

Also, normally the given graph is undirected, though we will sometimes impose a direction onto it if this helps for an algorithm.

3 Some problems

Now we give examples of some problems that will be studied, and how hard they are on the various graph classes.

- *Maximum Independent Set* (commonly called just Independent Set): An *independent set* of a graph $G = (V, E)$ is a subset I of the vertices such that no edge has both endpoints in I.

  It is well-known that independent set is NP-hard for general graphs. (See the textbook of Cormen et al. [CLRS00] for the proof that VertexCover is NP-hard. This also shows that Independent Set is NP-hard, because a set I is an independent set of a graph $G = (V, E)$ if and only if $V - I$ is a vertex cover.)

  As we will see, Independent Set can be solved efficiently (namely, in $O(m + n)$ time) on an interval graph. Likewise, it can be solved in linear time on a tree. On the other hand, Independent Set remains NP-hard for planar graphs.

- *Maximum Cut*: A *cut* in a graph $G = (V, E)$ is a partition of the vertices into two subsets $V = A \cup B$. More precisely, given such a partition, the cut is the set of edges with exactly one

[1] Obviously there are exceptions, for example edge coloring. So one should be a little careful, but for most problems that we'll see, assuming simplicity poses no difficulty.

endpoint each in A and B. The Maximum Cut problem is the problem of finding a partition with the maximum number of edges in the cut.

This problem is similar to the Minimum Cut problem, where we want to find the cut with the minimum number of edges. The Minimum Cut problem is well-known to be solvable in polynomial time (via Maximum Flow, see for examle [AMO93]). On the other hand, the Maximum Cut problem is known to be NP-hard (see e.g. [GJ79]).

As we will see, the Maximum Cut problem becomes polynomial if the graph is a planar graph. Also, the Maximum Cut problem is trivially solvable on trees: Since trees are bipartite graphs, simple use the partition from the bipartite graph; this will put *all* edges into the cut, which is clearly the best-possible. The complexity of Maximum Cut is open for interval graphs.

4 Typical questions

So the main question in this course is:

> Given a graph problem P and a graph class $\mathcal{C}$, how easy is it to solve P on a graph that belongs to $\mathcal{C}$?

Rather than studying this for each problem (as we did above), we will study this for each graph class. So given a graph class $\mathcal{C}$, typical questions that we ask are the following:

- Which graph problem becomes easier if the graph is in $\mathcal{C}$? (Thus, if the problem was NP-hard, does it become polynomial? If it was polynomial, can we decrease the running time?)
- Which graph problem is still NP-hard on graphs in $\mathcal{C}$?

The motivation for this type of questions is as follows: Assume you have graphs that arise from some application, and you notice that they appear to have some special structure. Then, if you happen to know that the problem you're trying to solve is easier on some graph classes, it would make sense to test whether your graphs actually belong to this class, and if so, run the faster algorithm for it.

This naturally raises a few more questions for a graph class $\mathcal{C}$:

- How easy is it to test whether a given graph G belongs to class $\mathcal{C}$? Is this NP-hard? Polynomial? Linear-time?
- What are equivalent characterizations of graphs in class $\mathcal{C}$?

  Quite often, one characterization of the graph class is easier to convert into an algorithm than another. For example, for *chordal graphs* (which are graphs without an induced 4-cycle; we will see them in a few lectures), the definition makes it easy to test chordality in $O(n^4)$ time. (There are better algorithms!) On the other hand, an equivalent characterization (a chordal graph is the same as a graph with a perfect elimination ordering) gives rise to a number of simple algorithms for chordal graphs.
- If G does not belong to $\mathcal{C}$, how "close" is G to being in $\mathcal{C}$?

  This is motivated by the hope of maybe modifying G into a graph G' in $\mathcal{C}$, solving the problem on G', and obtaining a solution (or at least an approximation thereof) for G from the solution of G'.

There are various ways of defining "close"; it could by deleting or adding vertices or edges, contracting or subdividing edges, replacing crossings by vertices ....

Sadly, these problems are just about always NP-hard.

References

[AGA99] Advanced Graph Algorithms. Lecture notes of a graduate course, University of Waterloo, Fall 1999.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[Bod93] Hans Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[CLRS00] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition.* MIT Press, McGraw-Hill Book Company, 2000.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[Gol80] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.

[NC88] T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms.* North-Holland, 1988.