Computer Science 331

Other Graph Problems and Algorithms

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #35

Outline

Other Graph Problems

Numerous other graph problems have applications involving scheduling and communication problems (among other things) and are therefore of interest

Some of these — which you might see in other courses and which would be studied in this course, if we had more time — are briefly introduced in these notes.

Single-Source Shortest Paths

We have already seen that *Dijkstra's algorithm* can be used to find a minimum-cost path from a given start vertex s to each vertex that is reachable from s, in a connected, weighted graph, if costs of all edges are greater than or equal to zero.

The problem is trickier if edge costs can be negative, because it is possible that there are *cycles* with negative costs, so that minimum-cost paths do not even exist!

The *Bellman-Ford* algorithm can be used to determine whether a given graph includes a negative-cost cycle; if the graph does not, then this finds minimum-cost paths (even when some edge costs are less than zero); it is slower than Dijkstra's algorithm when edge costs are all nonnegative.

All Pairs Shortest Paths

Now suppose that the input is a weighted undirected graph $G = (V, E)$ and we wish to find minimum-cost paths between *every* pair of vertices.

- If all edge costs are greater than or equal to zero then it is sufficient to visit each vertex s in turn run Dijkstra's algorithm with s as the start vertex.
- Another algorithm — the *Floyd-Warshall Algorithm* — can be used to solve this problem if some edge-costs are negative. This is generally faster than running the Bellman-Ford algorithm on every vertex.

Flow Networks

Motivating Question: *How much* water can pass through a river system, or through a network of pipes?

A *flow network* is a directed graph $G = (V, E)$ in which each edge (u, v) has a nonnegative *capacity* (max. rate of flow)

$$c(u, v) \geq 0$$

We assume that $c(u, v) = 0$ if $(u, v) \notin E$.

Each flow network also has a designated pair of district vertices:

- a *source* $s \in V$ (producing material)
- a *sink* $t \in V$ (consuming material "flowing" from the source)

Other vertices are "junctions" in the network.

Flows

A *flow* in G is a real-valued function

$$f : V \times V \to \mathbb{R}$$

that satisfies the following three properties:

- *Capacity Constraint:*

$$f(u, v) \leq c(u, v) \text{ for all } u, v \in V$$

- *Skew Symmetry:*

$$f(u, v) = -f(v, u) \text{ for all } u, v \in V$$

- *Conservation of Flow:*

$$\text{For all } u \in V \setminus \{s, t\}, \sum_{v \in V} f(u, v) = 0$$

Maximum Flows

The *value* $|f|$ of a flow f is defined as

$$|f| = \sum_{f \in v} f(s, v)$$

where s is the source vertex for the flow graph (as above).

Maximum Flow Problem: Given a flow network, compute a flow for this network with maximum value.

Several variations of the *Ford-Fulkerson Method* can be used to solve this problem efficiently.

Harder Problems

A variety of other graph problems are not believed to have asymptotically efficient (deterministic, worst-case polynomial time) solutions at all. Indeed, if $\mathcal{P} \neq \mathcal{NP}$ then none of these do.

Important examples:

- *Traveling Salesman Problem*

  Input: A weighted, connected, undirected graph `G = (V, E)` and the weights for its edges

  Problem: Find a *Hamiltonian cycle* — that is, a simple cycle that includes every vertex — whose cost is as small as possible

Harder Problems

- *Maximum Clique*

  Input: An undirected graph `G = (V, E)`

  Problem: Find a *clique* — a set of vertices $\mathtt{V}' \subseteq \mathtt{V}$ such each vertex $\mathtt{v} \in \mathtt{V}'$ is a neighbour of every *other* vertex $\mathtt{w} \in \mathtt{V}'$ — whose size is as large as possible.

- *Maximum Independent Set*

  Input: An undirected graph `G = (V, E)`

  Problem: Find an *independent set* — a set of vertices $\mathtt{V}' \subseteq \mathtt{V}$ such that no two vertices $\mathtt{v}, \mathtt{w} \in \mathtt{V}'$ are neighbours — whose size is as large as possible

Harder Problems

- *Graph Colouring:*

  Input: An undirected graph `G = (V, E)`

  Problems: Find a *colouring* — a map from the set `V` of vertices to a finite set of *colours*, such that no vertices `v` and `w` are mapped to the same colour if they are neighbours — that uses as few colours as possible

A variety of scheduling problems — including the problem of scheduling final exams so that students do not have time conflicts — can be modelled using this "Graph Colouring" problem

Continuations in Other Courses

The study of data structures and algorithms continues in other courses

- CPSC 335: *Information Structures II*

  Additional information about hash tables, search trees, sorting algorithms, and data structures and algorithms for other problems (including various computations on strings)

- CPSC 461: *Information Structures III*

  File structures, data structures and algorithms that can be used to store and manipulate extremely large sets of data, notably including data sets that are too large to be fit into main memory

Continuations in Other Courses

CPSC 413: *Design and Analysis of Algorithms I*. Includes

- Algorithm Analysis, including methods to deal with complicated expressions (including summations and recurrences) that you get when bounding the running times of more complicated iterative and recursive programs
- Algorithm Design Techniques: We have now seen algorithms that "Divide and Conquer," "Dynamic Programming," and "Greedy Algorithms."

  These are "algorithm design techniques." In CPSC 413 you learn (a bit) about how to use these to design algorithms of your own.
- Complexity Theory: Something that can, sometimes, be used to argue that problems *do not have* any efficient algorithms at all!

References:

- Cormen, Leiserson, Rivest and Stein
  Introduction to Algorithms (Second Edition)

  This is available online and in the library; it includes descriptions of all the algorithms mentioned in these notes, as well as chapters on the CPSC 413-related material mentioned above
- Wikipedia - while not *always* reliable, this seems to have quite a good set of pages about graph algorithms, as well as more general topics concerning algorithm design and analysis