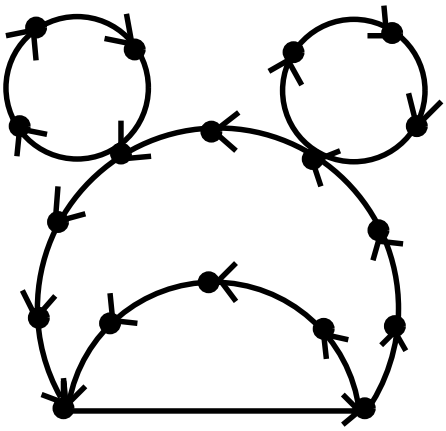

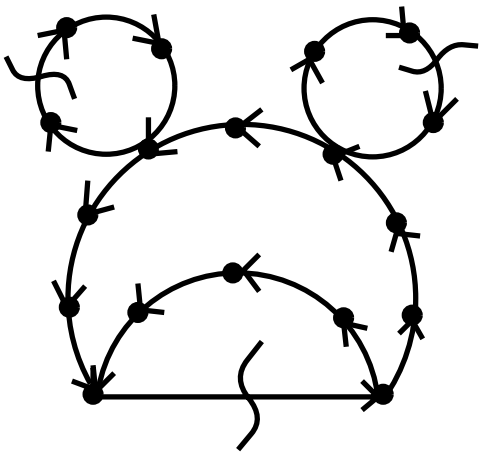

INPUT OUTPUT

16.11 Feedback Edge/Vertex Set

Input description: A (directed) graph $G = (V, E)$.

Problem description: What is the smallest set of edges E' or vertices V' whose deletion leaves an acyclic graph?

Discussion: Feedback set problems arise because many things are easier to do on directed acyclic graphs (DAGs) than general digraphs. Consider the problem of scheduling jobs with precedence constraints (i.e. , job A must come before job B). When the constraints are all consistent, the resulting graph is a DAG, and topological sort (see Section 15.2 (page 481)) can be used to order the vertices to respect them. But how can you design a schedule when there are cyclic constraints, such as A must be done before B, which must be done before C, which must be done before A?

By identifying a feedback set, we identify the smallest number of constraints that must be dropped to permit a valid schedule. In the *feedback edge* (or arc) set problem, we drop individual precedence constraints. In the *feedback vertex set* problem, we drop entire jobs and all constraints associated with them.

Similar considerations are involved in eliminating race conditions from electronic circuits. This explains why the problem is called "feedback" set. It is also known as the *maximum acyclic subgraph problem.*

One final application has to do with ranking tournaments. Suppose we want to rank order the skills of players at some two-player game, such as chess or tennis. We can construct a directed graph where there is an arc from x to y if x beats y in a game. The higher-ranked player *should* be at the lower-ranked player, although upsets often occur. A natural ranking is the topological sort resulting after deleting the minimum set of feedback edges (upsets) from the graph.

Issues in feedback set problems include:

- *Do* any *constraints have to be dropped?* – No changes are needed if the graph is already a DAG, which can be determined via topological sort. One way to find a feedback set modifies the topological sorting algorithm to delete whatever edge or vertex is causing the trouble whenever a contradiction is found. This feedback set might be much larger than needed, however, since feedback edge set and feedback vertex set are NP-complete on directed graphs.

- *How can I find a good feedback edge set?* – An effective linear-time heuristic constructs a vertex ordering and then deletes any arc going in the wrong direction. At least half the arcs must go either left-to-right or right-to-left for any vertex order, so take the smaller partition as your feedback set.

  But what is the right vertex order to start from? One natural order is to sort the vertices in terms of edge-imbalance, namely in-degree minus out-degree. Another approach starts by picking an arbitrary vertex v. Any vertex x defined by an in-going edge (x, v) will be placed to the left of v. Any x defined by out-going edge (v, x) will analogously be placed to the right of v. We can now recur on the left and right subsets to complete the vertex order.

- *How can I find a good feedback vertex set?* – The heuristics above yield vertex orders defining (hopefully) few back edges. We seek a small set of vertices that together cover these backedges. This is exactly a vertex cover problem, the heuristics for which are discussed in Section 16.3 (page 530).

- *What if I want to break all cycles in an undirected graph?* – The problem of finding feedback sets in undirected graphs is quite different from digraphs. Trees are undirected graphs without cycles, and every tree on n vertices contains exactly $n - 1$ edges. Thus the smallest feedback edge set of any undirected graph G is $|E| - (n - c)$, where c is the number of connected components of G. The back edges encountered during a depth-first search of G qualified as a minimum feedback edge set.

  The feedback vertex set problem remains NP-complete for undirected graphs, however. A reasonable heuristic uses breadth-first search to identify the shortest cycle in G. The vertices in this cycle are all deleted from G, and the shortest remaining cycle identified. This find-and-delete procedure is employed until the graph is acyclic. The optimal feedback vertex set must contain at least one vertex from each of these vertex-disjoint cycles, so the average deleted-cycle length determines just how good our approximation is.

It may pay to refine any of these heuristic solutions using randomization or simulated annealing. To move between states, we can modify the vertex permutation by swapping pairs in order or insert/delete vertices to/from the candidate feedback set.

Implementations: Greedy randomized adaptive search (GRASP) heuristics for both feedback vertex and feedback edge set problems have been implemented by Festa, et al. [FPR01] as Algorithm 815 of the *Collected Algorithms of the ACM* (see Section 19.1.6 (page 659)). These Fortran codes are also available from *http://www.research.att.com/∼mgcr/src/*.

GOBLIN (*http://www.math.uni-augsburg.de/∼fremuth/goblin.html*) includes an approximation heuristic for minimum feedback arc set.

The econ_order program of the Stanford GraphBase (see Section 19.1.8 (page 660)) permutes the rows and columns of a matrix so as to minimize the sum of the numbers below the main diagonal. Using an adjacency matrix as the input and deleting all edges below the main diagonal leaves an acyclic graph.

Notes: See [FPR99] for a survey on the feedback set problem. Expositions of the proofs that feedback minimization is hard [Kar72] include [AHU74, Eve79a]. Both feedback vertex and edge set remain hard even if no vertex has in-degree or out-degree greater than two [GJ79].

Bafna, et al. [BBF99] gives a 2-factor approximation for feedback vertex set in undirected graphs. Feedback edge sets in directed graphs can be approximated to within a factor of $O(\log n \log \log n)$ [ENSS98]. Heuristics for ranking tournaments are discussed in [CFR06]. Experiments with heuristics are reported in [Koe05].

An interesting application of feedback arc set to economics is presented in [Knu94]. For each pair A, B of sectors of the economy, we are given how much money flows from A to B. We seek to order the sectors to determine which sectors are primarily producers to other sectors, and which deliver primarily to consumers.

Related Problems: Bandwidth reduction (see page 398), topological sorting (see page 481), scheduling (see page 468).