SENSITIVITY ANALYSIS OF MINIMUM SPANNING TREES AND SHORTEST PATH TREES

Robert Endre TARJAN
Bell Laboratories, Murray Hill, NJ 07974, U.S.A.

Received 20 October 1981

Sensitivity analysis, shortest path, minimum spanning tree, path compression, graph algorithm, network optimization, transmuter

Let G be a graph, either undirected or directed consisting of a vertex set V and an edge set E. We shall use n to denote the number of vertices and m to denote the number of edges of G. Suppose each edge $e \in E$ has an associated real-valued *edge cost* c(e). A number of network optimization problems call for determining a subgraph of such a graph that is minimum with respect to some function of the edge costs. For example, the *minimum spanning tree problem* is that of determining, for a connected, undirected graph G, a spanning tree of minimum total edge cost. The *shortest path tree problem* is that of computing, for a given directed graph G and a given vertex r, a spanning tree rooted at r that contains a minimum-cost path from r to every other vertex. In such optimization problems it may be useful to measure the robustness of the solution. That is, given a minimum subgraph, we would like to know by how much we can perturb each edge cost individually without changing the minimality of the subgraph. We call this the *sensitivity analysis problem.*

Shier and Witzgall [3] have proposed several algorithms for sensitivity analysis of shortest path trees. Gusfield [2] has shown that two of their algorithms can be implemented to run in O(m log n) time and O(m) space, and has noted that his techniques also apply to sensitivity analysis of minimum spanning trees. In this paper we show how to perform sensitivity analysis of minimum spanning trees and shortest path trees in $O(m\alpha(m, n))$ time and O(m) space, where α is a functional inverse of Ackermann's function [4] defined as follows. For integers $i, j \geq 1$ we define A(i, j) by

$$A(1, j) = 2^j \qquad \text{for } j \geq 1;$$
$$A(i, 1) = A(i-1, 2) \qquad \text{for } i \geq 2;$$
$$A(i, j) = A(i-1, A(i, j-1)) \qquad \text{for } i, j \geq 2.$$

For integers m, n such that $m \geq n - 1 \geq 0$ we define $\alpha(m, n)$ by

$$\alpha(m, n) = \min\{i \geq 1 \mid A(i, \lfloor (m+1)/n \rfloor) > \log_2 n\}.$$

The algorithm uses path compression on balanced trees [6] and extends the algorithms of [6] for verifying and updating minimum spanning trees.

Let us begin by considering sensitivity analysis of minimum spanning trees. Suppose G is a connected, undirected graph and T is a spanning tree of G. The following lemma is well-known:

Lemma 1. T is a minimum spanning tree of G if and only if, for each non-tree edge f, the cost of f is at least as large as the cost of any edge on the (unique) simple path in T joining the ends of f. (In what follows we shall denote the path in T joining the ends of f by T(f).)

Suppose T is a minimum spanning tree of G. We wish to determine, for each edge e of G, by how much the cost of e can be changed without affecting the minimality of T. The following corollary of Lemma 1 provides the answer to this question.

Corollary 1. If f is a non-tree edge, T remains minimal until the cost of f is decreased by more than c(f) – c(e), where e is an edge of maximum cost on T(f). If e is a tree edge, T remains minimal until the cost of e is increased by more than c(f) – c(e), where f is a non-tree edge of minimum cost such that e lies on T(f).

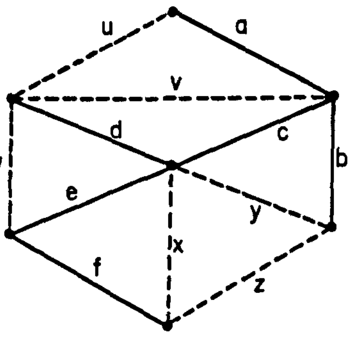
We can thus solve the sensitivity analysis problem for T by computing the increments and decrements specified in Corollary 1. To do this efficiently, we compute the increments and decrements for all the edges simultaneously using an auxiliary graph we call a *transmuter*.

A transmuter is a directed acyclic graph D that represents the set of fundamental cycles of G with respect to T, or more precisely the set of paths {T(f) | f is a non-tree edge}. We shall call the vertices of D *nodes* to distinguish them from the vertices of G. D contains one source (node of in-degree zero) s(e) representing each tree edge e, one sink (node of out-degree zero) t(f) representing each non-tree edge f, and an arbitrary number of additional nodes. The fundamental property of a transmuter is that there is a path from a given source s(e) to a given sink t(f) if and only if edge e is on tree path T(f). See Fig. 1.

Tarjan [6, section 7] describes a way to compute a transmuter containing O(mα(m, n)) nodes and edges in O(mα(m, n)) time. Given a transmuter, we can solve the sensitivity analysis problem for T in time linear in the size of the transmuter, as follows:

Step 1 (compute decrements for non-tree edges). Label each node of the transmuter with a real number by processing the nodes in topological order. To process a source s(e), label it with c(e). To process a node v that is not a source, label it with the maximum of the labels on its (immediate) predecessors. When the labeling is complete, each sink t(f) is labeled with the maximum cost of an edge on the tree path T(f). Thus

(a)



(b)

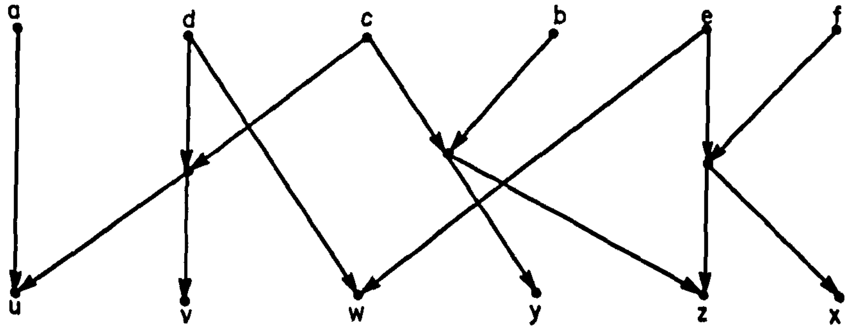


Fig. 1. A transmuter for a connected, undirected graph with spanning tree. (a) Graph. Tree edges are solid, non-tree edges are dotted. (b) Transmuter.

the decrement for f is its cost minus its label.

Step 2 (compute increments for tree edges). Label each node of the transmuter with another real number by processing the nodes in reverse topological order. To process a sink t(f), label it with c(f). To process a node v that is not a sink, label it with the minimum of the labels of its (immediate) successors. When the labeling is complete, each source s(e) is labeled with the minimum cost of an edge f such that e is on T(f). Thus the increment for e is its label minus its cost.

This algorithm requires not only $O(m\alpha(m, n))$ time but also $O(m\alpha(m, n))$ space to represent the transmuter. We can reduce the space to O(m) by processing the non-tree edges in groups. Let $m' = m - n + 1$ be the number of non-tree edges. Divide the non-tree edges into groups of size $O(m'/\alpha(m, n))$ and apply the above algorithm to each group (that is, to the subgraph consisting of the entire tree and the non-tree edges in the group). Each non-tree edge will receive its correct decrement, and each tree edge will receive $O(\alpha(m, n))$ increments, of which the minimum is the correct one. To process one group of non-tree edges takes

$$O\left(n + \frac{m'}{\alpha(m,n)}\,\alpha\left(\max\left\{n, O\frac{m'}{\alpha(m,n)}\right\}, n\right)\right) = O(m)$$

space and time by an extension [7] of the upper bound analysis in [4,6]. It follows that the total time to compute increments is $O(m\alpha(m, n))$, and the total space is O(m).

Transmuters can also be used to do sensitivity analysis of shortest path trees in $O(m\alpha(m, n))$ time and O(m) space. Let G be a directed graph and let T be a spanning tree of G rooted at a vertex r. The following lemma is well-known:

Lemma 2. For each vertex v in G, let d(v) be the total cost of the edges on the (unique) path from r to v in T. Then T is a shortest path tree if and only if $c(f) \geq d(w) - d(v)$ for every non-tree edge f = (v, w) in G.

Lemma 2 implies the following corollary, which Shier and Witzgall used to do sensitivity analysis of shortest path trees.

Corollary 2. Let T be a shortest path tree. If f = (v, w) is a non-tree edge, T remains a shortest path tree until the cost of f is decreased by more than $c(f) + d(v) - d(w)$. If e = (x, y) is a tree edge, T remains a shortest path tree until the cost of e is increased by more than $c(f) + d(v) - d(w)$, where f = (v, w) is a non-tree edge, such that w but not v is a descendant of y in T, which minimizes $c(f) + d(v) - d(w)$.

To analyze the sensitivity of a shortest path tree, we compute d(v) for every vertex in the tree. This takes O(n) time. Then we compute $\Delta(f) = c(f) + d(v) - d(w)$ for every non-tree edge f = (v, w). This takes O(m) time and gives us the decrements for the non-tree edges.

To compute the increments for the tree edges, we use a transmuter as follows: For every non-tree edge f = (v, w), we compute the nearest common ancestor *nca*(v, w) of v and w in T. This takes $O(m\alpha(m, n))$ time [1,6]. We define the *auxiliary edge* for a non-tree edge f = (v, w) to be f' = (nca(v, w), w). We construct a transmuter (with respect to the tree T) for the auxiliary graph G' = (V, E'), where $E' = \{e \mid e \text{ is a tree edge}\} \cup \{f' \mid f \text{ is a non-tree edge}\}$. Then we perform a backward labeling of the nodes of the transmuter (as in Step 2 of the sensitivity analysis method for minimum spanning trees), using $\Delta(f)$ as the label for sink t(f'). When the labeling is complete, each source is labeled by its increments.

This method requires a total of $O(m\alpha(m, n))$ time and space. As in the case of minimum spanning trees, the space bound can be lowered to O(m).

Transmuters have uses other than sensitivity analysis; they were originally developed for updating minimum spanning trees [6, section 7]. A lower bound result of Tarjan [5] implies that in the worst case a transmuter for an n-vertex, m-edge graph must have size $\Omega(m\alpha(m, n))$; thus devising a faster algorithm for sensitivity analysis seems to require a new idea.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, On computing least common ancestors in trees, SIAM J. Comput. 5 (1976) 115–132.

[2] D. Gusfield, A note on arc tolerances in sparse shortest path and network flow problems, unpublished manuscript (1981).

[3] D.R. Shier and G. Witzgall, Arc tolerances in shortest path and network flow problems, Networks 10 (1980) 277–291.

[4] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, J. ACM 22 (1975) 215–225.
[5] R.E. Tarjan, Complexity of monotone networks of computing conjunctions, Annals Discrete Math. 2 (1978) 121–133.
[6] R.E. Tarjan, Applications of path compression on balanced trees, J. ACM 26 (1979) 690–715.
[7] R.E. Tarjan, Worst-case analysis of set union algorithms, to appear.