On Scheduling Transmissions in a Network

Dan Gusfield

YALEU/DCS/TR 481

June 1986

Research funded in part by ONR grant N00014-82-K-0184

On Scheduling Transmissions in a Network

Dan Gusfield
Yale University, Department of Computer Science

1. Introduction

In the paper "Scheduling Transmissions in a Network", A. Itai and M. Rodeh [IR] give two algorithms to optimaly schedule, for a given objective function, the transmission of data from k nodes in a network to a single destination. The first algorithm runs in time $O(k|E||V|^2)$, and the second in time $O(k^2|E||V|\log|V|)$, where V is the node set and E is the edge set. The algorithms are complicated modifications of two particular known network flow algorithms.

In this note we point out that the transmission problem can be solved by performing at most k+1 network flow computations, where any network flow algorithm can be used. Hence the transmission problem can be solved in $O(k|V|^3)$ time for dense graphs, or $O(k|E||V|\log|V|)$ time for sparse graphs, using any of several well known network flow algorithms. This improves the two time bounds, gives a more direct and simpler solution to the problem, and permits the immediate application of any future improvements in network flow algorithms to speed up the solution of the transmission problem. The solution here is a direct application and extension of a result by H. Stone [S] concerning parametric network flow.

2. Problem Statement and Background

We refer to [IR] for the full formal model and generality of the transmission problem, and the formal definitions. In place of the general model, we illustrate the general problem with a special case that is easy to describe.

Graph G is a capacitated directed graph with nodes set V and edge set E, where k nodes of V are designated as emitters, and one node t of V is designated as the sink. Associated with each emitter v_i is a fixed number c_i. A number T is called *feasible* if it is possible to flow a total of $\sum_{i=1}^{k} c_i/T$ bits per time unit to the sink, where each emitter v_i puts out exactly c_i/T bits per time unit. That is, each emitter v_i will send exactly c_i/T bits per time unit in a continuous stream, along some fixed paths to the sink, and no edge will attempt to carry more than its capacity. Of

course, T will be infeasible below some value, and so the problem is to find the smallest feasible value of T. That is, find the smallest period T such that in every T time units, each emitter v_i emits exactly c_i bits in a continuous flow, and the bits from all the emitters flow to t without violating any of the edge capacity constraints.

In [IR] the above problem, as well as the full transmission problem defined in [IR], is shown to be solvable as the following parametric network flow problem. G is a directed graph with a source node s and a sink node t, with a directed edge of capacity c_i/T running from s to each of k nodes $v_1,...,v_i,...,v_k$ in V, where T is a parameter; each of the other edges has its own fixed capacity. The problem is to find the minimum value of T such that an s-t flow of value $\sum_{i=1}^{k} c_i/T$ is feasible, or equivalently, to find the minimum value of T such that the edges out of s form a minimum cut.

For any fixed value of T, the question of whether T is above, below or equal to the optimal value of T can be answered by a single network flow computation on G, and hence the transmission problem can be solved by some form of bisection search over the possible values of T. However, as pointed out in [IR], the number of tests in such methods depend on the size of $\sum_{i=1}^{k} c_i$, and hence the needed number of network flow computations would not be expressible in |V| and |E| alone. The bounds in [IR] are independent of $\sum_{i=1}^{k} c_i$, and their methods are not based on search over T. In this note we point out that efficient search over the feasible values of T is possible; the key theorem was developed in earlier applications of parametric network flow [S], and similar results appear in [ES].

2.1. Parametric Network Flow

Definition: Let H be a graph where every edge from source node s to node v_i has parameterized capacity $f_i(z)$, which is an *increasing* function of z; each other edge in H has its own fixed capacity. For an s-t cut C, let s(C) be the nodes on the s side of C and t(C) be the nodes on the t side of C. Note that either s(C) or t(C) alone define C.

The following theorem is the key to efficient search.

Theorem 1 [S]: If $z_1 < z_2$ then for any s-t cut C_1 in H which is minimum for $z = z_1$, there exists an s-t cut C_2 which is minimum for $z = z_2$, such that $s(C_1) \subseteq s(C_2)$. Hence there exists a sequence of no more than |V|-1 s-t cuts such that for any value of z, one of these cuts is a minimum cut for z.

This theorem was originaly stated only for increasing linear functions of z, but the proof in [S] actually shows the stronger result above.

3. Solving the Transmission Problem

In this section we apply Theorem 1 to the transmission problem.

Definition: Let S be the cut consisting of all the edges out of node s.

First, we transform the transmission problem by letting the parameter z replace 1/T. Then each edge in G from s to v_i has parameterized capacity zc_i, all other edges have their original constant capacites, and hence the transmission problem is to find the *maximum* value of z such that the minimum s-t cut has capacity $z\sum_{i=1}^{k} c_i$, i.e. so that the S is a minimum cut. Graph G now satisfies the conditions of Theorem 1, and we will use the theorem to bound the number of of minimum cuts that must be examined.

Defintion: Given parametrized graph G as above, let F(z) be the value of the maximum s-t flow as a function of z, and given an s-t cut C, let z(C) be the capacity of the cut C as a function of z.

Hence, for any s-t cut C, z(C) is a linear function of z, and F(z) is the lower envelope of the supperposition of the linear functions corresponding to all the s-t cuts. Hence,

Fact [ES], [G]: F(z) is a piecewise linear, convex, increasing function of z.

Defintion: The points where F(z) changes slope are called *breakpoints*.

At $z = 0$, S is a minimum s-t cut, and $z(S) = z\sum_{i=1}^{k} c_i$, so the transmission problem is really the problem of finding the first breakpoint of F(z) to the right of z=0. We call that breakpoint the *leftmost* breakpoint. See figure 1.

Note that the breakpoints of F(z) are well defined. Note also that Theorem 1 implies there can be at most |V|-1 breakpoints of F(z). To see this, let C_1 be an s-t cut which is minimum to the left of breakpoint z′, but not to the right of z′, and let C_2 be a cut which is minimum to the right of z′. By Theorem 1, we can assume that $s(C_1) \subseteq s(C_2)$. However, equality of those set would imply that the C_1 and C_2 are the same cut, which is impossible, since C_1 is not a minimum cut to the right of z′. Hence $s(C_1) \subset s(C_2)$, and it follows that F(z) has at most |V|-1 breakpoints. This can be improved as follows:

Lemma 1: For the transmission problem, F(z) has at most k breakpoints, and k+1 line segments.

Proof: Let C_1 and C_2 be two s-t cuts as above; then $s(C_1) \subset s(C_2)$, and C_1 and C_2 are minimum s-t cuts for different values of z. Consider the situation in figure 2, which shows C_1. A node which is connected to node s is called an s-node, and the others are called interior nodes. If $s(C_2) - s(C_1)$ contains only interior nodes, then $z(C_1) = q_1 + zw$, and $z(C_2) = q_2 + zw$, where w is the same constant for both. So either $z(C_1) = z(C_2)$, or these two linear functions never

intersect. They can't be the same, since then both cuts C_1 and C_2 would be minimum cuts in the same range of z. Hence $z(C_1)$ and $z(C_2)$ cannot intersect, so at least one of the cuts C_1 or C_2 will be a minimum s-t cut for no values of z. This contradicts the assumption, and so $s(C_2)$ - $s(C_1)$ contains an s-node, and since G has only k s-nodes, F(z) has at most k breakpoints. □

3.1. Efficient Search for the Leftmost Breakpoint

A method for finding all the breakpoints of any parametric network flow problem, when each edge capacity is a linear function of a commom parameter z, is given in [ES][1]. If there are k breakpoints, this method uses at most 2k+1 network flow computations. However, we will show below how to find the leftmost breakpoint using only k+1 network flow computations.

Definition: For any network flow problem with fixed capacities, if C and C′ are minimum s-t cuts, then C is said to be *left* of C′ if $s(C) \subseteq s(C')$. A minimum s-t cut which is to the left of all other minimum s-t cuts (for a problem with fixed capacities) is called *leftmost*.

Theorem 2 [FF]: For any network flow problem with fixed capacities, there always exists a leftmost minimum cut, and given a maximum s-t flow, it can be found in O(E) time.

For the transmission problem, if C_1 and C_2 are two s-t cuts, where C_2 is to the left of C_1, and $z(C_1) = q_1 + zw_1$, and $z(C_2) = q_2 + zw_2$, then $w_1 \leq w_2$. The importance of this in the transmission problem is that if, for a fixed value of z, there is more than one minimum cut, the leftmost minimum cut has the largest slope, so the minimum cut with largest slope can easily be found.

Algorithm for Finding the Leftmost Breakpoint

0. Let C be any s-t cut other than S.

1. Compute the intersection point, z^*, of z(S) and z(C),
   the linear capacity functions of the two respective cuts.

2. Fix the capacities of all the edges in G out of s by setting z to z^*.
   With these fixed capacities, compute the maximum flow and find the
   leftmost minimum s-t cut C^*.

3. If $F(z^*) = z^* \sum_{i=1}^{k} c_i$ (i.e. S is a minimum cut at z^*),
   then z^* is the leftmost breakpoint of F(z); terminate.

[1]Actually the method is very general and finds the breakpoints for any linear parameterized optimization problem.

4. Else, set C to C*, and go to step 1.

Figure 3 illustrates the working of the algorithm.

Theorem 3: The algorithm finds the leftmost breakpoint in at most k+1 iterations.

Proof: After every execution of step 2, C* is the s-t cut with largest slope that is a minimum cut for z^*. Hence the line segment of F(z) to the left of z^* is supported by z(C*), i.e. that part of F(z) runs along the line z(C*). If $F(z^*) \neq z^* \sum_{i=1}^{k} c_i$ in step 3, then $F(z^*) < z^* \sum_{i=1}^{k} c_i$, and $C \neq C^*$. Hence each iteration either terminates in step 3, or finds a new line segment of F(z). So termination occurs after at most k+1 iterations. When $F(z^*) = z^* \sum_{i=1}^{k} c_i$, z^* must be the leftmost breakpoint; S is a minimum s-t cut for both z=0 and $z=z^*$, and so it must be a minimum s-t cut for all values of z between them, but S is not minimum to the right of z^*, since $C \neq S$ has smaller capacity to the right of z^*. □

If an arbitrary minimum cut had been used in step 2, then 3k/2 iterations can be forced. When z^* is a breakpoint and C* is not the minimum cut of steepest slope, then it can take two iterations to discover a new edge of F(z). This can be made to happen for every other breakpoint.

4. Acknowledgement

I thank Baruch Awerbuch and Andrew Goldberg for bringing this problem to my attention, and Baruch Awerbuch for suggesting the transformation $z = 1/T$, which simplified an earlier exposition.

5. References

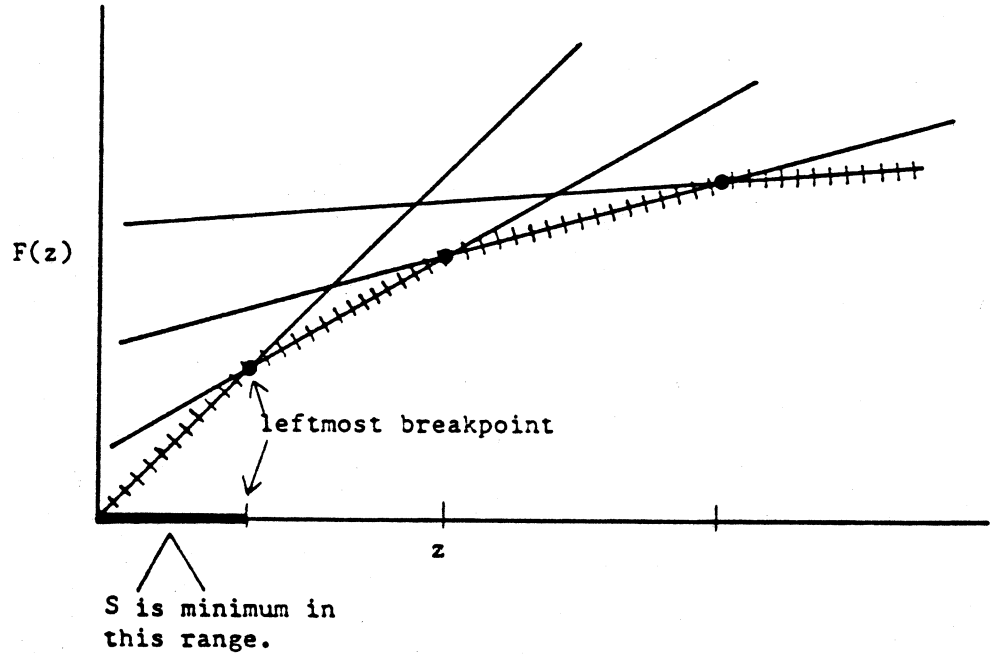
[ES] M. Eisner and D. Severence. Mathematical Techniques for the Efficient Record Segmentation in Large Shared Databases. JACM, vol. 23, no. 4 (1976).

[FF] L. Ford and D. Fulkerson. Flows in Networks. Princeton press, 1962.

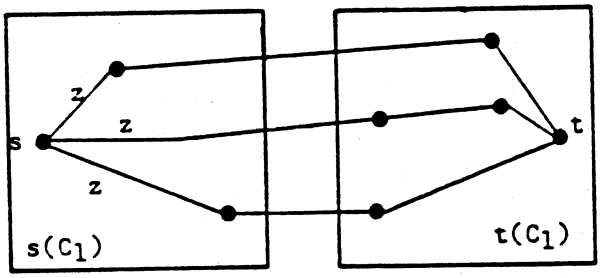
[G] D. Gusfield. Parametric Combinatorial Computing and a Problem in Program Module Distribution. JACM vol. 30 no. 3, July 1983, pp. 551-563.

[IR] A. Itai, and M. Rodeh. Scheduling Transmissions in a Network. Journal of Algorithms 6, 409-429 (1985).

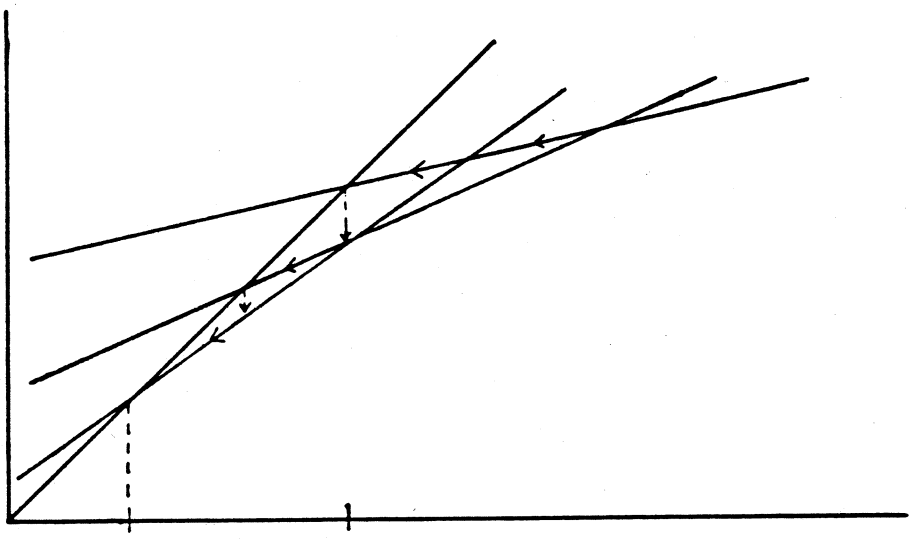
[S] H. Stone. Critical Load Factors in Two-Processor Distributed Systems. IEEE Transactions on Software Engineering, vol. se-4, no. 3, May 1978.



1. F(z) is shown with a hatched line. There are three break points on F(z).



2. cut C_1.



leftmost breakpoint found in three network flow computations.

initialize z*

3.