ROUTING-SEQUENCE OPTIMIZATION FOR CIRCUIT-SWITCHED NETWORKS

Fu Chang

__Fu Chang__ is a member of technical staff in the Switched Network Design Department at Holmdel, New Jersey. He works on optimal design and survivability problems of digital switched networks. He has a Ph.D. in mathematical statistics from Columbia University. He joined AT&T in 1984.

A new routing design algorithm is presented for circuit-switched networks with dynamic routing capabilities. It is assumed that, in such networks, a transaction request can be set up by trying a sequence of paths connecting the originating switch and destination switch. The request will be connected through the first path that has an available circuit at the time of the request. The objective of this algorithm is to maximize the network throughput or revenue under a given capacity and load. The algorithm is a heuristic that first solves a simpler linear programming (LP) problem. It then transforms the LP solution into a feasible solution for the original nonlinear optimization problem. The solution is then improved through "route trimming" and "route expansion" processes. By using variants of Karmarkar's method implemented on the AT&T KORBX® system to solve the LP problem, the algorithm can be applied to large networks with potentially hundreds of switches. Furthermore, in many cases the algorithm provides near-optimal routing solutions with objective values that are within 0.5 percent of the optimal solutions.

Introduction

This paper describes a routing sequence optimization (RSO) algorithm that provides a routing scheme for a circuit-switched network with dynamic routing capability under given offered loads and given link capacities.

We assume that the circuit-switched network is supported with the features required for a sequential routing scheme. That is, a trans-

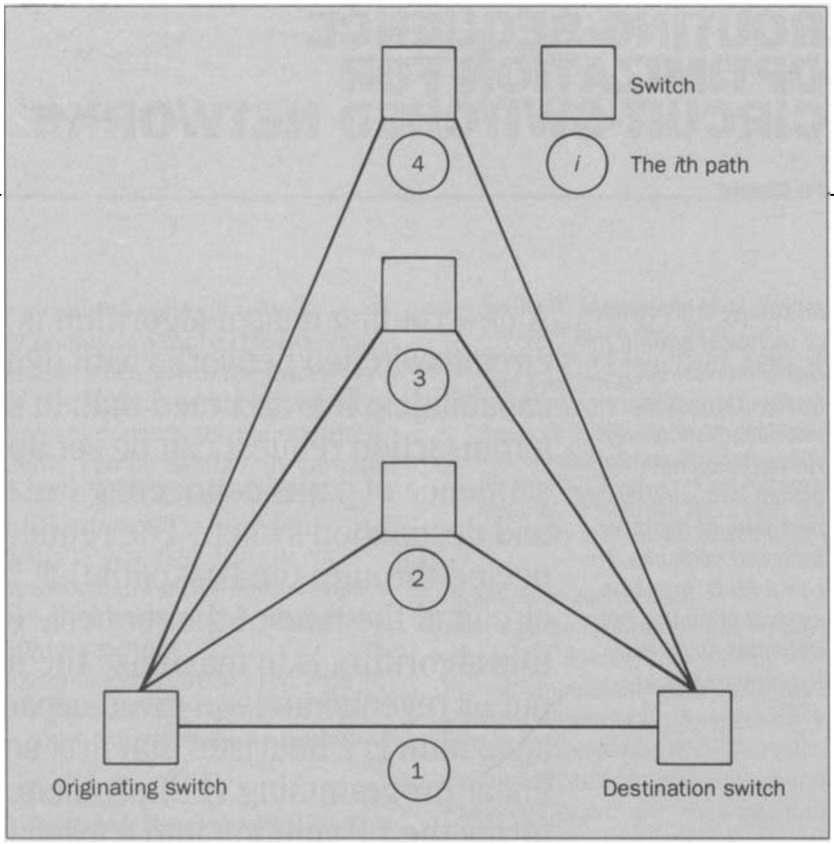

Figure 1. Routing sequence for circuit-switched networks.

action request can be set up from the originating switch to the destination switch by trying a sequence of paths connecting the two switches (see Figure 1). The request will be connected and routed through the first path within the sequence that is found to have free capacity at the moment the call request is made. The purpose of the algorithm is to provide a sequence of paths, called a *route*, for each node pair so that the average network throughput or the revenue generated by the throughput is maximized.

The routing design problem is a nonlinear optimization problem. The RSO algorithm, however, is based on solving an approximate problem using linear programming and then improving the derived solution to achieve a better objective value (throughput or revenue). In many examples on which we tested our method, we have seen that the RSO algorithm can achieve objective values which are very close to the optimal values. A related algorithm that uses a linear programming approach provides a network capacity design method for networks with dynamic routing.[1]

Formulation of the Problem

Given a network consisting of N nodes, the number of node pairs is $N(N-1)/2$. We assume that for each node pair k there is a link (also labeled k) whose capacity is C_k trunks. We also assume that the demand load for node pair k is D_k erlangs with peakedness Z_k.

A network uses paths to transport traffic. Different node pairs use different paths to transport their traffic. A path for a node pair k can be either the direct path (link k) or a two-link path connecting the same endpoints. A route $R(k)$ for node pair k is a sequence of paths $p_1, p_2, \ldots, p_{J(k)}$ for k such that the traffic blocked from p_j is overflowed to p_{j+1}, for $j = 1, 2, \ldots, J(k) - 1$. Note that the total number of paths $J(k)$ varies with k. A routing scheme is a set $\{R(k)\colon k = 1, 2, \ldots, K\}$ consisting of routes for all node pairs, where $K = N(N-1)/2$.

Given a path p for node pair k, the load f_p that p carries for k is called the *path flow* of p. If we sum up all flows over the paths containing the same link l, the sum is called the *link flow* of l. Neither a path flow nor a link flow can be calculated independently of any other path flows or link flows. Rather, one has to solve a set of nonlinear equations to obtain all path and link flows simultaneously. It suffices to say that for given D_k, C_k, Z_k, and $R(k)$ for all k, one can derive f_p for all $p \in R(k)$, $k = 1, 2, \ldots, K$, by solving a set of nonlinear equations. We refer to these equations as *flow equations*. The formulation of these equations is based on queueing approximations, similar to those described in Chapter 4 of Cooper.[2] Similar flow equations also were solved in Ash et al.[1]

The routing design problem is to find a routing scheme that maximizes the network throughput

$$\sum_{k=1}^{K} \sum_{p \in R(k)} f_p \qquad (1)$$

A similar problem is to find a routing scheme that maximizes the revenue

$$\sum_{k=1}^{K} r_k \sum_{p \in R(k)} f_p \qquad (2)$$

where r_k is the revenue per unit of load for node pair k. In fact, if we set r_k to 1 for all k, then problem (2) reduces to problem (1). Thus, the throughput maximization problem is a subproblem of the revenue maximization problem.

Solution Process

For the size of the problem that we are dealing with, the computational requirements make it virtually impossible to always find an optimal solution. A heuristic approach must be taken as the practical way to solve our problems.

We divide our heuristic approach into three steps:

1. *Initialization*: generating candidate paths for each node pair.
2. *Linear programming (LP) optimization*: setting up and solving an approximate LP problem.
3. *Nonlinear approximation*: deriving a solution for the nonlinear problem and further improving it.

A flow chart of our algorithm is given in Figure 2. In the following sections, we provide the detailed descriptions of these steps.

Initialization

Candidate paths are selected by examining certain transmission quality assurance constraints. To find paths satisfying the constraints requires checking every possible path, composed of one or two links, by various standards, such as distance restrictions, avoidance of using certain nodes as via nodes, etc. We retain only those paths meeting the set of constraints.

Linear Programming Approximation

The objective of the LP problem is the same as the original problem, that is, to maximize the throughput

$$\sum_{k=1}^{K} \sum_{p \in R(k)} f_p \qquad (3)$$

or the revenue

$$\sum_{k=1}^{K} r_k \sum_{p \in R(k)} f_p \qquad (4)$$

The LP problem concerns itself only with the candidate paths in a route, without considering the order in which these paths are attempted. Once the flows f_p are determined, the sequence of paths within each route is set by the nonincreasing order of path flows.

When the LP problem is formulated initially, $R(k)$ in problem (3) or (4) is simply taken to be the set of candidate paths for k, as is generated in the initialization step. In subsequent iterations, when a new LP problem is formulated, $R(k)$ will be taken as the paths within the route for k generated in the previous nonlinear approximation step.

The constraints for the LP problem are specified

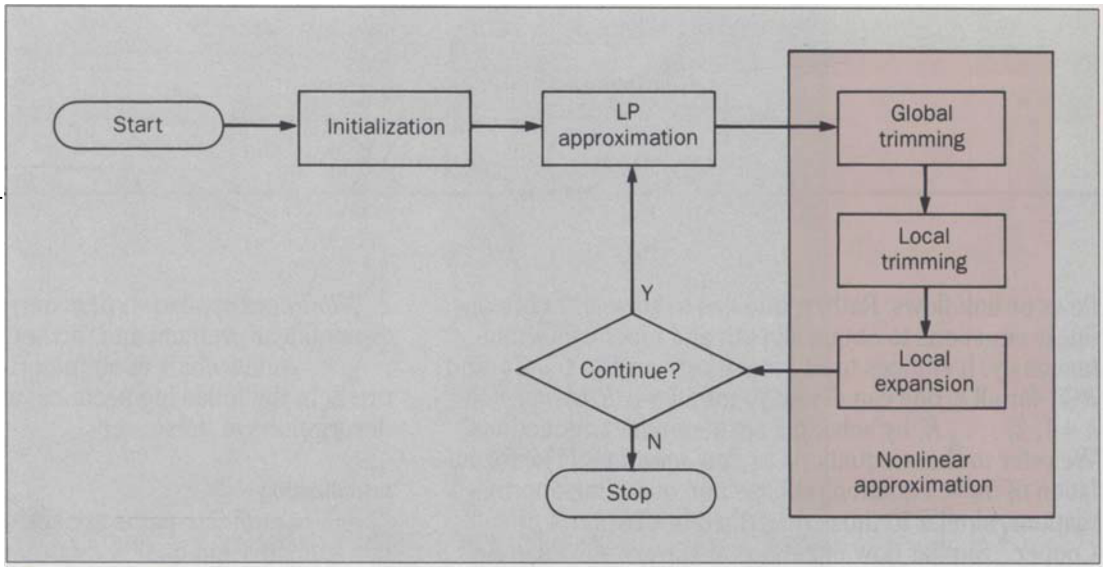

Figure 2. Flow chart for routing sequence optimization.

as follows. A set of inequalities specifies the constraints on link flows:

$$\sum_{k=1}^{K} \sum_{\substack{p \in R(k) \\ p \supset l}} f_p \leq C_l \qquad \text{for all } l \qquad (5)$$

Here the notation $p \supset l$ means that path p contains link l. Thus, constraint (5) requires that the sum of flows over the paths containing link l should not exceed the capacity of link l.

A second set of inequalities specifies the lower and upper bounds on the route flows:

$$A_k \leq \sum_{p \in R(k)} f_p \leq D_k \qquad \text{for all } k \qquad (6)$$

where $0 \leq A_k \leq D_k$.

Here, A_k is taken to be the carried load for k in the previous feasible solution for the nonlinear approximation step. When the LP problem is formulated initially, we set up a routing scheme using simple heuristic rules. Let A_k be derived from the solution to the flow equations associated with this routing scheme. In a later step, when a new LP problem is formulated, the routing scheme will be just the one generated in the previous step. The reason for introducing A_k in constraint (6) is to assure some minimal throughput for each node pair k. Without such lower bounds, it is possible that some of the node pairs become virtually disconnected under an overloaded situation, if maximizing throughput or revenue is the only objective.

The final constraint consists of lower and upper bounds for path flows specified by the following inequalities:

$$0 \leq f_p \leq u_p \qquad \text{for all } p \in R(k) \text{ and all } k \qquad (7)$$

When the LP problem is formulated initially, u_p is taken to be the capacity of p, which is the minimum capacity of the links composing path p. In later iterations, when a new LP problem is formulated, we use the previous offered load to constrain the path flow so that the LP problem becomes a closer approximation to the original problem. The offered load to a path is defined to be

$$\frac{g_p}{1 - b_p}$$

where g_p is the path flow for p determined by the routing scheme generated in the previous (nonlinear approximation) step and b_p is the path blocking of p.

To summarize, LP flow optimization has the objective of maximizing expression (3) or (4) under constraints (5), (6), and (7). The formulation of the original nonlinear problem specifies the same constraints as the LP problem at the first iteration with the addition of the flow equations. Hence, any feasible solution of the original problem is also feasible for the initial LP problem. This is not true at subsequent iterations because of the tightening of the upper bounds u_p.

Nonlinear Approximation

Having obtained the LP solution, we can sort the available paths for each node pair according to nonincreasing path flows. The paths receiving zero flows are discarded. The ordered sequences of the remaining paths are then used to form a routing scheme. We then solve the flow equations for this routing scheme to obtain new path flows. Next, we attempt to improve this solution by the following operations: (1) global trimming, (2) local trimming, and (3) local expansion. We call the entire procedure a ***nonlinear approximation.***

Global Trimming. Each step of global trimming involves eliminating one or more of the least-used paths in each route. At the end of each step, the network throughput (or revenue) is reevaluated, by solving the flow equations, to determine whether the objective value has been improved. If so, another step of global trimming is initiated.

Global trimming limits the number of paths in all routes simultaneously and thereby achieves a better objective value in a very efficient and parallel operation. The next operation has the same purpose but evaluates routes individually.

Local Trimming. After the completion of global trimming, we examine the benefit of deleting the last path from a single node pair. Since there exist as many routes as node pairs, it would be extremely time-consuming to solve the flow equations to assess the value of each local trimming. On the other hand, since the perturbation of a local trimming operation is not likely to be large, its value can be assessed on the basis of the following approximation.

To examine whether a local trimming operation is profitable, we start with the route for the first node pair ($k = 1$). Any path with zero flow is deleted immediately from the route. Let s denote the last path with nonzero flow, and let l (and m, if there are two links) denote the link associated with it. The loss of flow caused by the deletion of s from $R(1)$ is approximately

$$\text{Loss} = f_s \tag{8}$$

which is just the path flow of s before the deletion.

The load offered to link l before the deletion is

$$O_l = \sum_{k=1}^{K} \sum_{\substack{p \in R(k) \\ p \supset l}} f_p/(1 - b_p) \tag{9}$$

where b_p is the blocking probability of path p. After the deletion, the total load offered to l is reduced to approximately

$$O_l' = O_l - \frac{f_s}{1 - b_s} \tag{10}$$

where b_s is the blocking of s. Thus, the new blocking for link l after the deletion of s is approximately

$$B_l' = \beta(O_l', C_l) \tag{11}$$

where $\beta(O, C)$, the Erlang-B formula, denotes the blocking when a load O is offered to a link with capacity C.[2,3]

The new blocking B_m for link m, if it exists, can be approximated in a similar way. From the new blockings, we can calculate the gain of flow for each route $R(k)$ $(k \neq 1)$ due to the deletion of s. Now we sum the gain of flow over all the routes and subtract equation (8) from this sum to obtain the net gain of flow caused by the deletion of s. Naturally, if the net gain is found to be negative, then we do not delete s. Otherwise, we make the deletion, and adjust offered load and link blocking on the affected links according to equations (10) and (11) for the subsequent approximation. We then proceed to approximate the net gain caused by the deletion of the last path of the second route, etc.

Once we run through all the routes and decide for each route whether to delete the last path, we have obtained a new routing scheme. At this point, we solve the flow equations for this scheme to determine whether it has achieved a better objective value as expected. If not, we restore the previous solution and stop. Otherwise, we proceed to apply local trimming once again to the new routing scheme just obtained. To summarize:

1. For each route, approximate the net gain in the objective value caused by the deletion of its last path. If the net gain is positive, delete the path and update offered load and blocking on the affected links; otherwise, do not delete the path.
2. Solve the flow equations for the new routing scheme obtained in step 1. If the objective value has been improved, return to step 1. Otherwise, restore the previous solution and stop.

Local Expansion. Local expansion is similar to local trimming, except that a path is added instead of being deleted.

We start with the first route ($k = 1$). Again, any path with zero flow is deleted immediately from the route. To select the candidate path to add, we list all paths connecting the first node pair and satisfying the transmission quality requirements. We pick the path that has the highest connectivity. (The connectivity of a path is one minus its blocking probability.)

The gain of flow t from using this extra path is approximately

$$\text{Gain} = o_t(1 - b_t) \tag{12}$$

where o_t is the traffic that overflowed from all paths of $R(1)$ before path t was added to $R(1)$ and b_t is the blocking of t. Let l denote one of the links associated with t. The load offered to link l before the expansion of $R(1)$ is assumed to be O_l. Then the load offered to l after the expansion is approximately

$$O_l' = O_l + o_t \tag{13}$$

Thus, the new blocking for link l is approximately

$$B_l' = \beta(O_l', C_l) \tag{14}$$

The remaining steps for computing the net gain are similar to those described above for local trimming. Thus, to summarize,

1. For each route, approximate the net gain in the objective value caused by the addition of a path. If the net gain is positive, add the path and update the offered load and blocking on the affected links; otherwise, do not add the path.
2. Solve the flow equations for the new routing scheme obtained in step 1. If the objective value has been improved, return to step 1. Otherwise, restore the previous solution and stop.

When all the above operations have been completed, we compare the final objective value with the LP optimum, established in the previous step. If the former exceeds α percent of the latter (we used $\alpha = 99.5$), then we stop. If not, we return to the LP approximation with reduced upper bounds for all path flows. If at some later iteration of the nonlinear approximation, the objective value is less than that obtained at the previous iteration, then we restore the previous solution and stop the algorithm.

Performance of the RSO Algorithm

If the RSO algorithm stops at the end of the first iteration of the nonlinear approximation step, we can safely say that the algorithm achieves a solution whose objective value is very close to the true optimum [within $(1 - \alpha)$ percent]. The reason is as follows.

Let the optimal value of the original nonlinear problem (1) [or (2)] be V_{ORG}, let that of the corresponding LP problem be V_{LP}, and let the objective value reached at the end of the first iteration of the nonlinear approximation step be V_{HEU}. Then the following inequalities hold:

$$V_{\mathrm{HEU}} \leq V_{\mathrm{ORG}} \leq V_{\mathrm{LP}}$$

The right inequality is true because the optimal solution for the original problem is a feasible solution for the corresponding LP problem. Now, if we stop at the first iteration of the nonlinear approximation step, the heuristic objective value V_{HEU} must be very close [within $(1-\alpha)$ percent] to the LP optimum V_{LP} by the stopping criterion. From the above relationships, we conclude that the heuristic objective value is also close to the original optimum.

In testing the above algorithm on practical problems, we observed that, for most of our examples, the algorithm stopped at the end of the first iteration. The only cases in which the algorithm needed more iterations were those in which the congestion became so serious that some node pairs had end-to-end blockings above 80 percent. In such cases, we do not know how close our solution is to the optimal one. However, our solutions were still favorable compared to other solutions determined by heuristic methods.

We have implemented the RSO algorithm on the AT&T KORBX® system[4] in FORTRAN. To solve our problems of small and midrange size (with network sizes of up to 70 nodes), we used the dual power series method in the KORBX system. For the larger problem (corresponding to network size of about 100 nodes) we used the dual conjugate gradient method. We have solved many routing design problems in our test examples. The largest LP problem we solved had about 10,000 constraints and 70,000 variables. The entire RSO algorithm for the same problem took about 90 minutes of central processing unit (CPU) time. For heavily loaded networks, it took more CPU time to solve the routing design problem, because multiple iterations were needed to reach the final solution and each iteration took a longer time than other networks would take.

Conclusion

We have presented a routing algorithm for circuit-switched networks with dynamic routing capabilities. The algorithm uses linear programming to derive an initial routing solution and then improves the solution by some nonlinear-based heuristics. By using Karmarkar's algorithm implemented on the KORBX system to solve the LP approximation problem, we have found that the RSO algorithm is computationally feasible for very large routing problems. Furthermore, in most of the examples, the algorithm provided near-optimal solutions; they are at most 0.5 percent below the optimal objective value. The only solutions that we could not prove to be near-optimal are those in which the network was heavily loaded. However, experiments show that the algorithm still provides good solutions in those cases, compared to solutions that were obtained by other heuristic methods.

References

1. G. R. Ash, R. H. Cardwell, and R. P. Murray, "Design and Optimization of Networks with Dynamic Routing," *Bell System Technical Journal*, Vol. 60, No. 8, October 1981, pp. 1821-1845.
2. R. B. Cooper, *Introduction to Queueing Theory*, 2nd edition, North Holland Publishing, New York, 1981.
3. D. L. Jagerman, "Methods in Traffic Calculations," *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 7, October 1984, pp. 1283-1310.
4. Y. C. Cheng, D. J. Houck, Jr., J. M. Liu, M. S. Meketon, L. Slutsman, R. J. Vanderbei, and P. Wang, "The AT&T KORBX® System," *AT&T Technical Journal*, Vol. 68, No. 3, May/June 1989.

(Manuscript received February 1, 1989)