Copyright © 1981 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 8, October 1981
Printed in U.S.A.

Design and Optimization of Networks With Dynamic Routing

By G. R. ASH, R. H. CARDWELL, and R. P. MURRAY

(Manuscript received March 6, 1981)

The growth of electronic switching systems and the high-capacity interoffice signaling network provide an opportunity to extend telephone network routing rules beyond the conventional hierarchy. Network models are described that illustrate the savings inherent in designing networks for dynamic, nonhierarchical routing. An algorithm for engineering such networks is discussed, and the comparative advantages of various path-routing and progressive-routing techniques are illustrated. A particularly simple implementation of dynamic routing called two-link dynamic routing with crankback is discussed and is shown to yield benefits comparable to much more complicated routing schemes. The efficient solution of embedded linear programming (LP) routing problems is an essential ingredient for the practicality of the design algorithm. We introduce an efficient heuristic optimization method for solution of the LP routing problems, which greatly improves computational speed with minimal loss of accuracy. We also project computational requirements for a 200-node design problem, which is the estimated size of the intercity Bell System dynamic routing network in the 1990s.

I. INTRODUCTION AND SUMMARY

The rapidly growing stored program control (SPC) network, consisting of electronic switching systems interconnected by common-channel interoffice (CCIS) signaling links, provides a significant opportunity to extend the telephone network routing rules beyond the conventional hierarchy. In the SPC network, there are no restrictions to hierarchical route choices or to routing rules which remain fixed in time, but we may rationally consider network configurations which use dynamic, nonhierarchical routing (DNHR). The term dynamic describes routing techniques which are time-sensitive, as opposed to present-day hierarchical routing rules which are time-fixed. An important variable in the dynamic routing strategy is the frequency with which network routing rules are updated.

1.1 Savings possibilities with dynamic routing

There are two major opportunities to improve the planned network design (forecast) with more advanced routing techniques. First, because of its fixed nature, present hierarchical routing cannot really take much advantage of load variations which arise from business/ residence, time zones, seasonal variations, and other reasons. By allowing time varying, or dynamic routing, some of this penalty can be reduced. Second, the present hierarchical routing has rigid path choices, plus low blocking on final links which limit flexibility and reduce efficiency. If we choose paths based primarily on cost and relax the present rigidity in network structure, a more efficient network should result. The upper limits on improvement in these two areas are discussed first.

1.1.1 Noncoincidence effects

It is estimated from a 28-node intercity network model (Fig. 1) that about 20 percent of the network's first cost can be attributed to designing for time varying loads using our present static hierarchical routing techniques. To show this, we first designed a hierarchical network using a conventional cluster busy-hour approach. Then, to quantify the extra capacity being provided, we also designed the 28-node model for the individual hourly loads. These hourly networks were obtained by using each hourly load, and ignoring the other hourly loads, to dimension a hierarchical network that would perfectly match that hour's load. This procedure results in 17 separate network designs, one for each hour.

Figure 2 is a plot of the normalized network cost (including switching and facility cost) required for the cluster busy hour and hourly network designs. On the top line, the cluster busy-hour solution had a network capital cost of one unit to satisfy all 17 hours of load with fixed, hierarchical routing. The 17 hourly networks, shown on the lower curve, represent the normalized capital cost of the circuit miles and trunks actually required at each hour to satisfy the load. Three network busy periods are visible: morning, afternoon, and evening. We can also see a noon-hour drop in load, and an early-evening drop as the business day ends and residential calling begins in the evening. The hourly network curve separates the capacity provided in the cluster busy-hour solution into two components: below the curve is the capacity actually needed at each hour to meet the load; above the curve is the capacity which is available but is not needed at that hour. This additional capacity exceeds 20 percent of the total network capacity

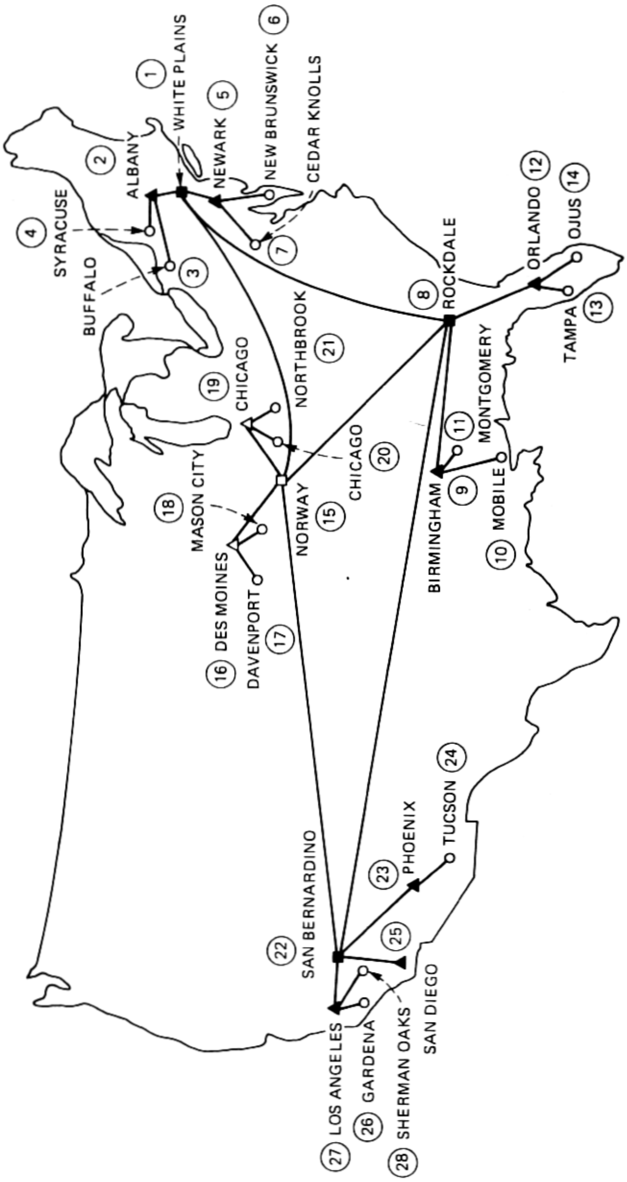


Fig. 1—Intercity network model for 28-node network. (Dark nodes indicate 10-node model.)

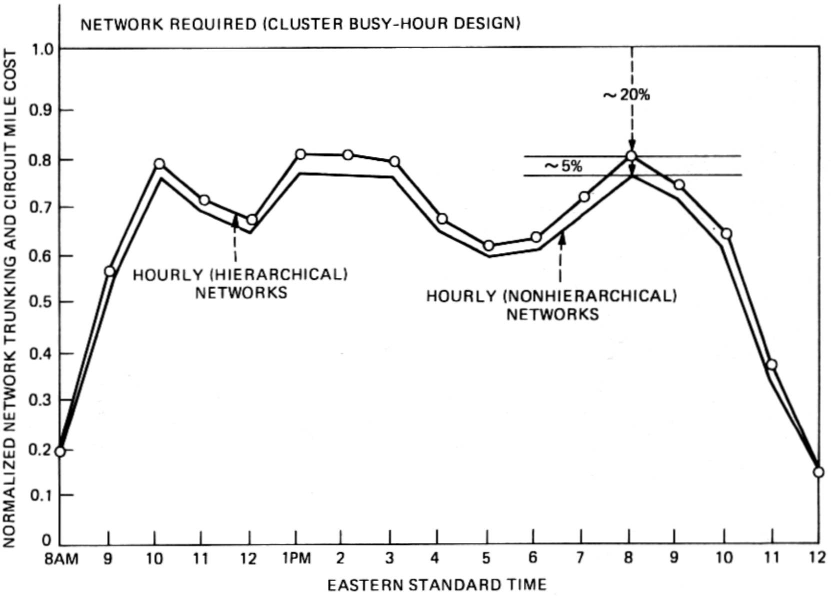


Fig. 2—Network first cost for 28-node network.

through all hours of the day. This gap represents the capacity put in the network to meet noncoincident loads, and suggests a maximum limit on network reduction which might be achieved through improved routing techniques.

1.1.2 Limited path selection effects

Additional benefits can be provided in network design by allowing a more flexible intercity routing plan that is not restricted to hierarchical routes. Our approach allows the selection of shortest (nonhierarchical) paths. Applied to each hourly load, this approach yields an overall savings of about 5 percent in comparison to the hierarchical hourly networks. Figure 2 also displays these results and shows that the 20 percent bound discussed above has increased to a total of 25 percent. This additional savings potential translates into actual benefits by introducing nonhierarchical shortest path routing into the design, as is done in the DNHR network design algorithm.

Figure 3 illustrates the limitation that the hierarchy imposes in the 28-node network between San Diego and Birmingham. The alternate paths between these points go through two regional centers, San Bernardino, Ca. and Rockdale Ga., providing relatively long paths. Selecting more direct paths, for example the Tucson, and Phoenix, Az. and Montgomery, Al. paths, would provide design benefits. Allowing the optimum choice of intercity routes beyond the hierarchical choices (i.e., nonhierarchical networks) yields design savings. This includes allowing the present final paths to use alternate routing, which in many cases would further improve the network efficiency.

1.2 Summary

In Section 3.2, we describe the route formulation of the unified algorithm (UA). In this formulation the allowed traffic patterns (routes) are formed for each point-to-point demand prior to traffic assignment in the routing optimization step. Three routing methods are considered in designing networks using the route formulation method:

(*i*) Progressive routing in which a call progresses through the network one switch at a time without retracing its path until it either reaches its destination or arrives at an intermediate switch from which it has no outlet.

(*ii*) Multilink path routing in which a call blocked by a busy trunk group on a path may use the capabilities of the SPC network to be "cranked back" to the originating node and attempt the next path in the route.

(*iii*) Two-link path routing, which is identical to multilink routing, except that a path from origin to destination may have at most two links.

We find that design savings on the order of 10–15 percent are possible when using these routing methods as compared to present hierarchical techniques. From the savings results and implementation considerations, we conclude that two-link routing is preferred.

We next consider another formulation of the UA called the path formulation, which is specifically tailored to examine two-link routing options. This method does not preselect allowable routes, but allows the traffic allocation step to assign traffic directly to paths in order to minimize network cost. Routes are formed after the optimization step to realize the desired flows. A flow feasibility algorithm is described

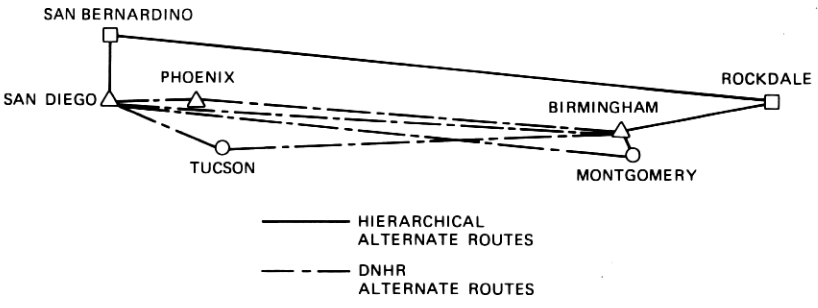


Fig. 3—Shortest path choice.

which forces the resulting path flows to be realizable. Three two-link routing methods for realizing the optimum path flows are then considered, varying in complexity from a very flexible method, CGH routing (developed by Chung et al)[1], to a very simple method called sequential routing. The latter method consists of offering all traffic to an ordered list of two-link paths with the overflow from one path being offered to the next path; the ordered list may change by time-of-day to take advantage of traffic noncoincidence.

We find that the routing techniques investigated using the path formulation achieve at least 1–2 percentage points additional savings over the routing techniques studied using the route formulation. We then find that sequential routing incurs an insignificant cost penalty when compared to more flexible routing schemes and, because of its simplicity, we conclude that sequential routing is the preferred routing method.

Efficient optimization techniques are considered in Section IV. These methods allow the design of very large networks for dynamic routing using reasonable computer resources. Finally, potential Bell System applications are discussed in Section V.

II. DYNAMIC ROUTING CONCEPTS: DESIGN, SERVICING, AND CONTROL

Figure 4 illustrates the three primary components of the network design and administration functions as three interacting feedback loops around the network. The network offered load is shown to consist of predictable, average demand components, unknown forecast errors, and day-to-day variation components. The feedback controls function to regulate the service provided by the network through capacity and routing adjustments. Network design (or planned servicing) operates over a year-long interval, drives the network capacity expansion, and preplans routing patterns to minimize network costs. Demand servicing accounts for the existing capacity and, on a weekly basis, fine-tunes link sizes and routing patterns to account for forecast errors inherent in the year-long design loop. Real-time control makes limited adjustments to the preplanned routing patterns to account for normal daily shifts in load patterns.

Network provisioning for dynamic routing depends primarily on performing off-line calculations for network design and demand servicing. The off-line calculations select the optimal routing patterns from a very large number of possible alternatives in order to minimize the trunking network cost. The term dynamic routing frequently suggests an extensive search for the optimal routing assignment to be performed in real time. This extensive search is in fact being made but most of the searching is performed in advance using an off-line design system

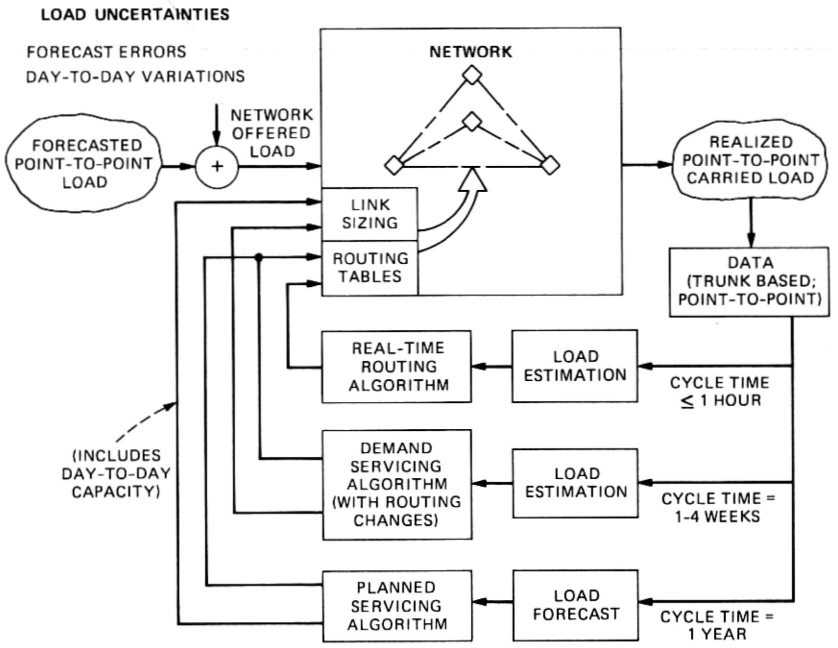

LOAD UNCERTAINTIES
FORECAST ERRORS
DAY-TO-DAY VARIATIONS
FORECASTED POINT-TO-POINT LOAD
+
NETWORK OFFERED LOAD
NETWORK
LINK SIZING
ROUTING TABLES
REALIZED POINT-TO-POINT CARRIED LOAD
DATA (TRUNK BASED; POINT-TO-POINT)
REAL-TIME ROUTING ALGORITHM
LOAD ESTIMATION
CYCLE TIME ≤ 1 HOUR
(INCLUDES DAY-TO-DAY CAPACITY)
DEMAND SERVICING ALGORITHM (WITH ROUTING CHANGES)
LOAD ESTIMATION
CYCLE TIME = 1-4 WEEKS
PLANNED SERVICING ALGORITHM
LOAD FORECAST
CYCLE TIME = 1 YEAR

Fig. 4—Planned servicing, demand servicing, and real-time control as interacting feedback loops around the network.

and an off-line demand servicing system. The effectiveness of the design depends on how accurately we can forecast the expected load on the network. Errors associated with the forecast are corrected in the demand servicing process described in the companion article.[2] The only routing decisions necessary in real time involve conditions that also become known in real time: day-to-day load variations, network failures, and network overloads. Procedures for real-time routing are also described in the companion article.

III. DESIGN ALGORITHM

3.1 Overview

In this section, we describe the algorithm used to design near minimum cost nonhierarchical networks using dynamic routing. This algorithm is termed UA because it combines into one systematic procedure various network design concepts, such as

(*i*) Using time-sensitive dynamic routing to take advantage of traffic noncoincidence,

(*ii*) Routing traffic along the least costly paths,

(*iii*) Favoring large, more efficient trunk groups, (*iv*) Using efficient trunk group blocking levels determined by the economic hundred call seconds (ECCS) method, and

(*v*) Minimizing incremental network cost.

The first two concepts were described in Section 1.1. A brief description of the other three concepts incorporated in the UA is given below.

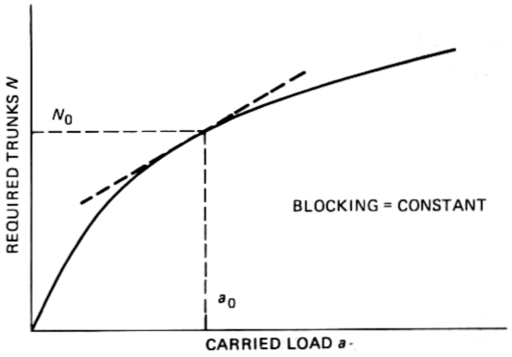
3.1.1 Favoring large trunk group

Figure 5 illustrates the number of trunks, N, required to carry a particular carried load, a, at constant blocking. From the shape of the curve comes the well-known fact that at constant blocking the number of additional trunks required to carry an increment of offered load decreases as the trunk group size increases. Hence, it is advantageous to combine several traffic parcels into one large parcel to be routed over a large trunk group since one large trunk group is inherently more efficient than several smaller trunk groups.

In the UA, larger trunk groups are favored through the use of a link incremental cost metric proportional to the slope $(\partial N/\partial a)$ of the trunks versus load curve. Thus, the link metric indicates the attractiveness of this link to carry additional traffic.

3.1.2 Use efficient blocking levels

Figure 6 illustrates the cost trade off between carrying traffic on the direct trunk group between A and B, and the alternate network that overflow calls will use. The problem is to find the optimum value of blocking (or, equivalently, the number of trxnks) to handle the offered load at a minimum cost. This question was first answered by Truitt[3] who derived the concept of an ECCS based on the direct path to



LINK METRIC $M_l \propto \frac{C\partial N}{\partial a}$, WHERE C = COST PER TRUNK

Fig. 5—Efficiency of large trunk groups.

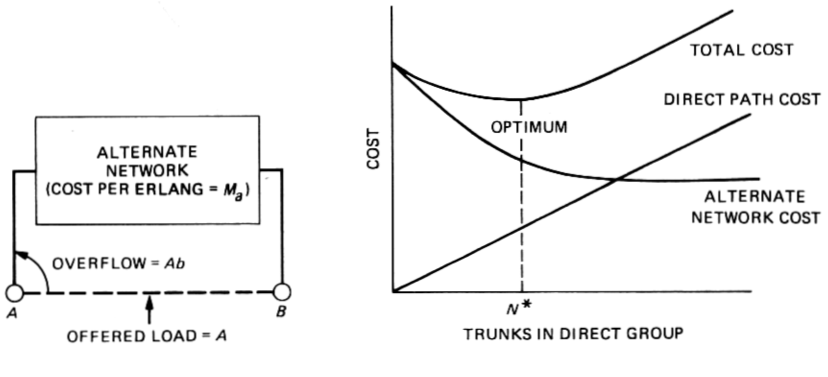


Fig. 6—Optimum blocking (ECCS method).

alternate path cost ratio and the marginal capacity of the alternate path. Truitt's ECCS method is commonly used today in both intercity and metropolitan network design. This method is also used in the UA.

3.1.3 Minimize incremental network cost

Network cost and performance are nonlinearly related. Hence, the network design problem is inherently a nonlinear programming problem. To avoid the complexities associated with nonlinearity, the network cost function can be linearized around the present operating point and the linearized (incremental) cost function minimized to yield a minimum cost network.

This approach of minimizing the incremental network cost has been successfully used by other investigators. Yaged[4] has used this technique to find a near minimum cost facility network to satisfy trunk demands when the facility links display a concave facility cost versus channel capacity relationship. For his problem, Yaged demonstrated that this technique satisfied the Kuhn-Tucker conditions which are necessary (but not sufficient) for optimality. An analogous approach was used by Knepley[5] who applied the minimal incremental cost concept to the design of the automatic voice network (AUTOVON).

Figure 7 shows the iterative loop for the route formulation of the UA. Basic input parameters include trunk cost, point-to-point offered loads, and required point-to-point grade-of-service (GOS).

The router finds the shortest paths (sequences of links) between points in the network. Using assumed link blocking levels, the router then forms the paths into candidate routes (sequences of paths) and determines the proportion of flow appearing on each path in the route for each unit of offered load. This method of forming routes from

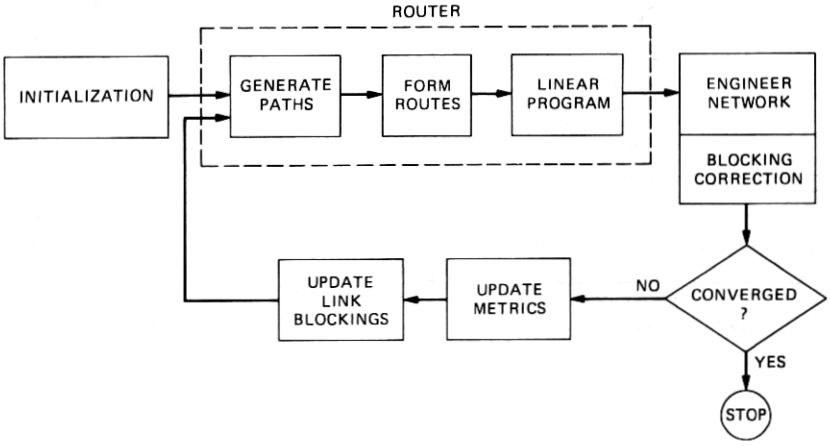


Fig. 7—Unified algorithm iterative loop.

assumed link blockings is a key feature of the UA. It eliminates the nonlinear relation between link blocking, number of trunks, and offered load from the optimization step, and it also permits investigation of a wide variety of routing schemes.

The LP then assigns flow to the candidate routes to minimize network cost. The output from the router is the optimum routing plan consisting of the routes to be used in each hour. This routing is provided to the engineering program which determines the flow on each link and sizes the link to meet the design level of blocking used in the router step. Once the groups have been engineered, the cost of the network can be evaluated and compared to the last iteration.

If the network cost is still decreasing, the update module (*i*) computes the slope of the capacity versus load curve on each link and updates the link cost using this slope as a weighting factor, and (*ii*) computes a new level of link blocking using the ECCS method. The new link lengths and blockings are fed to the router which again selects shortest paths, and so on.

3.2 Detailed description

3.2.1 Initialization

An initial set of link blockings and metrics are calculated based on the ECCS method. Initial link blockings are determined assuming that the overflow path is the shortest two-link path between the endpoints with a marginal capacity of 28 CCS.

3.2.2 Router

The router consists of both a route generator and an LP. The route generator constructs a set of candidate routes for each point-to-point demand pair in each design hour. Each route candidate contains just enough paths to meet the GOS constraint. The LP then selects which routes will be used in each hour and in what proportion.

Since the method of constructing routes depends on the routing discipline (progressive, multilink, or two-link) to be used, we defer the discussion of how these various routes are formed to their respective sections. For now we assume that the route generator forms the proper number of routes for each demand pair, and calculates the portion of route carried load on each link for the routing discipline used. The operation of the UA is such that almost any routing scheme can be used, merely by using the proper route generator.

The second step in the router is the LP, which assigns the offered traffic to the candidate routes in order to minimize the network incremental cost.

First, we introduce the following notation:

L = number of links.
K = number of demand pairs.
H = number of design hours.
J_k^h = number of routes for demand pair k in hour h.
P_{jk}^{ih} = proportion of carried load on route j for point-to-point demand pair k on link i in hour h.
M_i = incremental link cost metric in terms of dollar cost per erlang of carried traffic for link i.
R_k^h = offered load to demand pair k in hour h.
r_{jk}^h = carried load on route j of demand pair k in hour h.
A_i^h = offered load to link i in hour h.
a_i = maximum carried load on link i over all hours.
g_{jk}^h = route blocking on route j of demand pair k in hour h.
b_i^h = blocking on link i in hour h.

Then the LP will select the r_{jk}^h and the resulting a_i so as to minimize

$$\sum_{i=1}^{L} M_i a_i$$

subject to

$$\sum_{k=1}^{K} \sum_{j=1}^{J_k^h} P_{jk}^{ih} r_{jk}^h \le a_i \qquad i = 1, 2, \cdots, L \qquad h = 1, 2, \cdots, H$$

$$\sum_{j=1}^{J_k^h} \frac{r_{jk}^h}{1 - g_{jk}^h} = R_k^h \qquad h = 1, 2, \cdots, H \qquad k = 1, 2, \cdots, K$$

$$r_{jk}^h \ge 0, \; a_i \ge 0.$$

Inputs to the LP are P_{jk}^{ih} and g_{jk}^h from the route generator, M_i from the previous metric calculation, the link blockings b_i^h, and the R_k^h. Outputs from the LP are the r_{jk}^h, the assignment of carried load to the routes, and a_i, the associated link capacity (maximum carried load). In many cases, the IBM LP package (MPSX-370) was used to obtain the results reported here. All point-to-point traffic was first assigned to its least expensive route to form a feasible solution to the LP; this solution was used as a starting basis. In those cases where a heuristic optimization method (HOM) (see Section IV) is used to solve the LP, the output will be a set of r^h_{jk}, which approximates the optimal route flows.

3.2.3 Network engineering

After the LP has assigned traffic to routes, the network must be engineered to achieve a link blocking no higher than the assumed blocking used as input to the router. In this way, the GOS constraint will be satisfied, or at least the GOS will be no worse than that calculated by the router. If the GOS is not satisfactory, it is corrected by the blocking correction algorithm described below.

To arrive at a consistent set of hourly blockings and offered loads, an iteration scheme is used. The iteration uses the present estimates of the link offered loads to size each link in its peak hour and calculate blocking estimates in side hours. After all groups have been sized, new proportions of carried load are calculated using the blocking estimates and the routing pattern given by the LP. The link flows are then recalculated and the process repeated. The iteration is continued until the sum of the absolute blocking changes is less than a prescribed convergence threshold. Engineering can be accomplished either by using a single parameter traffic model or a two-parameter traffic model. Results given in this article are for the single-parameter case. Fractional trunks were allowed so as to achieve the required blocking exactly. This stabilizes the iterative loop and speeds convergence.

3.2.4 Blocking correction algorithm

If a route blocking in the engineered network exceeds a threshold, the blocking on the first path is decreased until the route blocking is equal to the desired GOS. The additional traffic which must be carried to reduce the route blocking to the desired GOS will, thus, be carried on the path which has the minimum incremental cost, and the network cost increase required to correct the route blocking should be close to minimal.

Once an engineered network solution is obtained, the route blockings needing correction are rank ordered and the highest route blocking is corrected first. After the new link blockings are obtained, routes are once again checked for blocking violations and the entire process is repeated until an engineered network solution is found which does not violate the route blocking constraint. The blocking correction has been made part of the engineering loop as shown in Fig. 7.

3.2.5 Calculation of new metrics

The expression for the link metric is $C_i \partial N_i / \partial a_i$, or the cost per trunk multiplied by the rate of change of trunks required to keep the blocking constant with a changing carried load. Hence, this is the incremental cost to carry an increment of load at constant blocking on link i. In particular, the partial derivative is approximated by

$$M_i = \frac{C_i[N_i(a_i + \Delta a_i) - N_i(a_i)]}{\Delta a_i},$$

where

C_i = cost of one trunk on link i
$N_i(a)$ = trunks required on link i for carried load a (for the link blocking b_i)
Δa_i = incremental carried load (normally set to 5 percent of a_i)

3.2.6 Calculation of more efficient blockings

The ECCS approach of Truitt[3] is used (Fig. 6) to calculate ECCS values in the UA. The objective is to calculate the number of trunks, N^*, (and, hence, the link blocking, B) that will minimize the total cost of carrying load A over the combination of the direct path and the alternate paths. To do this the network cost is first written as:

$$\begin{aligned} \text{Cost} &= CN + \alpha M_a \\ &= CN + AbM_a, \end{aligned} \qquad (1)$$

where α is the overflow load from link AB ($\alpha = Ab$) and M_a is an equivalent metric for the alternate route network. A partial derivative is taken of eq. 1 with respect to N and the resulting expression set equal to zero to obtain the minimum.

3.3 Candidate routing methods

3.3.1 Progressive routing

Progressive routing is familiar since the Bell System hierarchy is an example of progressive routing. In this scheme, when a call is sent from one node to another node, the control of the call is also passed to the next node. No crankback to a previous node is allowed, but the call must continue toward its destination at each stage, or be blocked. The main difficulty with progressive routing is to avoid looping. In the hierarchy this is prevented automatically by the structure of the network. In our nonhierarchical design, the assumption was made that the history of the call could be carried via CCIS. In that way, the electronic switching processor would know the nodes to which the call had already been routed, and disallow them as the next outlet choice.

Besides preventing looping, route control is also used to promote efficient trunk use. Basically, we prohibit excessive alternate routing which can result in calls routing on paths with many links, thus "stealing" trunks from calls which can complete on one or two links. This situation has a cascading effect and can result in inefficient trunk use, with fewer call completions than otherwise possible. To promote efficient trunk use, we eliminate paths with a large number of links which are unnecessary to meet the required GOS.

In the dynamic version of progressive routing, traffic is allocated to the most economical next node choices on a time varying basis.

3.3.1.1 Route proportions and blocking. A simple example of the computation of route blocking and proportions is given in Fig. 8. From the assumed blocking on each link and the progressive routing pattern, the load offered to, and overflowing from, each link is calculated. From this information, the route blocking and proportions are determined.

3.3.2 Multilink path routing

Path routing implies selection of an entire path between points in the network before a connection is actually provided on that path. If a connection on one link in a path is blocked, the call then seeks

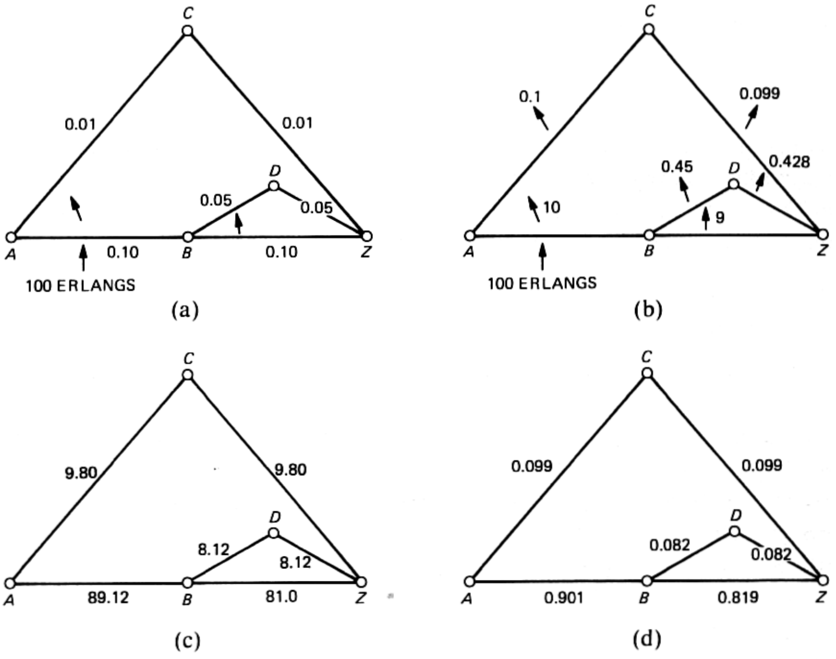


Fig. 8—Example of progressive route proportions. (a) Routing and link blocking. (b) Overflow loads (total carried load = 98.9 erlangs) (c) Link-carried loads. (d) Link proportions.

another complete path. Implementation of such a routing technique could be done through control from the originating office, plus a multiple link crankback capability to allow paths of greater than two links to be used. Path-to-path routing is nonhierarchical, and allows the choice of the most economical paths rather than being restricted to hierarchical paths.

Dynamic path routing is achieved by allocating fractions of the traffic to routes, and allowing the fractions to vary as a function of time. To generate more than one route for each point-to-point pair, one approach is to use cyclic routing. This method has as its first route $(1, 2, \cdots, M)$, where the notation (i, j, k) means all traffic is offered first to path i, which overflows to path j, which overflows to path k. The second route of the cyclic router is a cyclic permutation of the first route: $(2, 3, \cdots, M, 1)$. The third route is likewise $(3, 4, \cdots, M, 1, 2)$ and so on. This approach has computational advantages because its cyclic structure requires considerably fewer calculations to find the proportions for all routes than does a general collection of paths. The route blockings of cyclic routes are identical; what varies from route to route is the proportion of flow on the various links.

3.3.2.1 Route proportions and blocking. Figure 9 illustrates that some links may be common to more than one path and, hence, route blocking calculations and route carried flow calculations can become involved. From the assumed blocking on each link and the path-to-path routing pattern, the load offered to, and overflowing from, each link is calculated and from this information the route blocking and proportions are determined. More complicated routes are handled by a method given in Ref. 6.

3.3.3 Two-link path routing

In the design of multilink path networks, about 98 percent of the traffic was routed on one- and two-link paths even though paths of greater length were allowed. Because of switching costs, paths with one or two links are usually less expensive than paths with more links. Therefore, two-link path routing was introduced and uses the greatly simplifying restriction that paths can be two links in length at most. It requires only single-link crankback to implement and uses no common links, but is otherwise identical to the multilink scheme. It achieves nearly the same network savings as multiple-link path routing, and appears to be very attractive as a network routing alternative. Computation of route proportions is greatly simplified for two-link routing, since common links cannot occur on one route.

3.4 Route formulation results and conclusions

We consider here the cost of a 10-node subset of the 28-node network

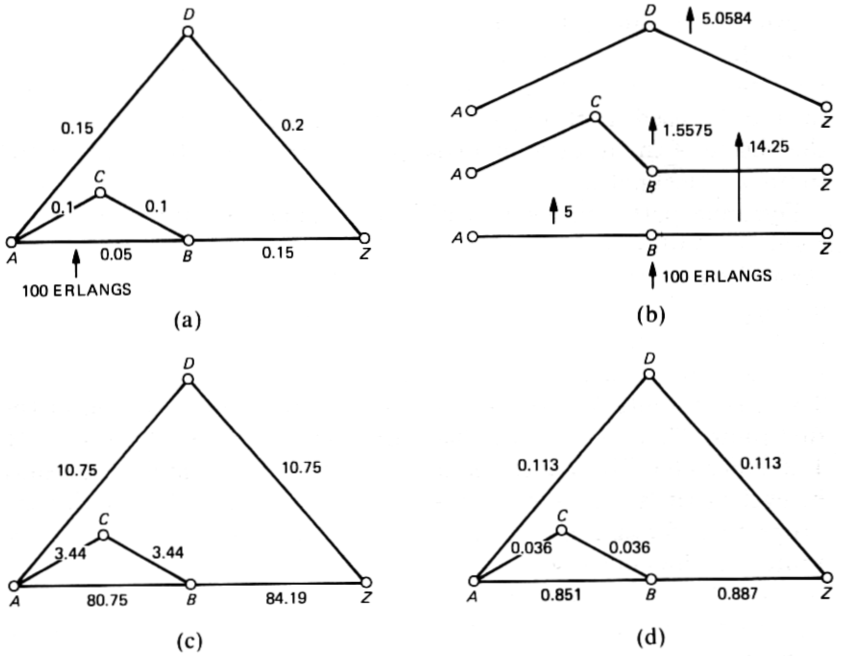


Fig. 9—Example of multilink proportions. (a) Link blockings: routing is $ABZ \rightarrow ACBZ \rightarrow ADZ$. (b) Path overflow loads and blocking: carried load = 10.749 erlangs and blocking = 0.32 for path ADZ; carried load = 3.4425 erlangs and blocking = 0.3115 for path $ACBZ$; carried load = 80.75 erlangs and blocking = 0.1925 for path ABZ. (c) Link-carried loads; route carried load = 94.94 erlangs. (d) Link proportions.

(Fig. 1) designed for multihour loads. Results for large networks are in general agreement with these results. We illustrate designs for hierarchical, progressive, multilink, and two-link networks to satisfy the traffic loads for a single hour of load and also for three network busy hours (10 a.m., 1 p.m., and 8 p.m.). The 10-node hierarchical networks were designed using current standard practices. In the design of DNHR networks, the IBM mathematical programming system, MPSX-370, was used to solve the necessary LP in the multihour design and was run to optimality in each iteration (this is feasible for the 10-node network problem). The GOS objective was 0.005 blocking, and five routes were allowed for each point-to-point demand pair in each hour.

3.4.1 Ten-node single hour results

The UA can design a network for a single hour simply by assigning all the traffic for a particular point-to-point pair to the least expensive route for that pair. There is no need to generate more than one route for each point-to-point pair since the direct route is the least expensive in the single-hour case.

Table I gives single-hour network design results using the 10 a.m.

load data, together with the percent savings for progressive routing, multilink routing, and two-link routing in comparison to the network engineered for hierarchical routing. The average network point-to-point GOS is also shown for each network design. The UA design cost usually converged in about five iterations. The savings for progressive routing and two-link routing are only slightly smaller than multilink routing. The average network GOS for the DNHR networks were all better than the hierarchy.

The primary reasons that the UA can save about 6–7 percent over a hierarchical design appear to be that (I) the UA has a better choice of routing, and (II) all groups can be sized for an efficient blocking level. In the 10-node network, the algorithm used paths from Los Angeles, Ca. to Orlando, Fl. that passed through Birmingham, Al. and Phoenix, Az., along with the more normal paths through San Bernardino, Ca. and Rockdale, Ga. used by the hierarchical design. Additionally, no existing final groups were sized for one percent blocking, hence, the average trunk occupancy was higher. For example, the Rockdale to White Plains, N.Y. group was sized for 16 percent blocking by the UA, and paths through the subtending sectional centers (in the hierarchy) were used to carry traffic overflowing the Rockdale–White Plains group so that the overall point-to-point blocking objective was met. In fact, the average blocking on groups that would be interregional finals in a hierarchy was about 21 percent in the UA. This resulted in higher occupancy of these expensive interregional groups.

3.4.2 Ten-node multihour results

We now discuss hierarchical, progressive, multilink, and two-link networks to satisfy the traffic loads for three network busy hours (10 a.m., 1 p.m., and 8 p.m.). From the results in Table II we conclude that there is little difference in potential network cost savings between progressive routing, two-link routing, and multilink routing. In fact, it appears that using CCIS crankback and originating node control will only save about an additional one percent in network cost. The reason is that most traffic in the various dynamic routing networks is routed on the same links, because for many point-to-point pairs, these routing

Table I—Single-hour unified algorithm results for 10-node network (10 a.m. load)

Network Routing	Cost	GOS	Savings (%)
Hierarchical	$5,949,500	0.009	
Progressive	5,567,800	0.004	6.4
Multilink	5,511,100	0.005	7.4
Two-link	5,555,900	0.005	6.6

Table II—Network designs for 10-node network (based on three hours)

Network Routing	Cost	Savings (%)	Network (GOS)	Hour
Hierarchical	$7,160,000			
Progressive	6,043,100	15.6	0.003	10 a.m.
			0.002	1 p.m.
			0.003	8 p.m.
Multilink	5,980,100	16.5	0.002	10 a.m.
			0.001	1 p.m.
			0.002	8 p.m.
Two-link	6,064,300	15.3	0.003	10 a.m.
			0.002	1 p.m.
			0.003	8 p.m.

methods carry a significant amount of traffic on the direct path and on the same two-link, first-alternate path.

Because progressive routing, two-link routing, and multilink routing designs are very close in cost, the preferred routing method should be based on ease of implementation. Progressive routing requires a history of visited nodes to be sent with each call to prevent looping. Since no central point has complete control of a particular call, it would also be quite difficult to measure point-to-point blocking. We contrast this to the use of originating node control in multilink or two-link routing which makes it easier to measure point-to-point blocking. The blocking measurement is necessary for network servicing in order to adjust routing and augment trunk groups to satisfy unforeseen loads; the blocking measurement would indicate when corrective action is necessary. Having originating node control of every call is also helpful for real-time routing, which attempts to maximize use of the network in the face of unusual load conditions. For a description of servicing and real-time routing, see Ref. 2. On the basis of these implementation considerations and the comparable savings, two-link routing appears to be the preferred routing method.

3.5 Path formulation

As explained earlier, the route formulation decided on the possible routes a call may take prior to the LP assigning traffic to the candidate routes at minimum cost. The choice of routes was limited because of the large number of candidates. For example, the number of routes that can be formed from ten paths is 10!, or over 3 million routes. Hence, the restricted choice of routes could result in suboptimality, since a better route not contained in those generated may exist.

The path formulation forms routes after the optimization step. Hence, the LP and "form routes" blocks would be interchanged in Fig. 7. The LP assigns traffic directly to the candidate paths at minimum cost.

The first step in the router stage is to generate the required number of one- and two-link paths. These paths are then passed to the LP, which is somewhat different in structure than that of the route formulation. This difference arises since the amount of flow that can be carried on a particular path depends on the blocking on that path and on the flow assigned to all other paths comprising the particular route. For instance, if the blocking on a path were 20 percent and the offered load were 100 erlangs, it would be impossible to carry more than 80 erlangs on this path. Hence, some method is needed to determine upper limits on path flow so that the resulting flows selected by the LP are feasible. Such questions of feasibility did not arise in the route formulation since the link blocking probabilities were embedded in the link proportions.

3.5.1 Flow feasibility algorithm

An iterative method of using upper bounds to force flow feasibility is shown in Fig. 10. Here we incorporate flow feasibility constraints into the router stage. Immediately after the generation of paths, initial upper bounds on path flows are set for use by the first LP iteration. At this point, nothing is known about the amount of flow which is optimal on any path. Hence, we desire to constrain the LP as little as possible. For this reason, the initial upper bound on flow on any path j for demand pair k is set according to the following formula:

$$UPBD_{jk} = R_k(1 - B_{jk}),$$

where

$UPBD_{jk}$ = upper bound on flow on path j of demand pair k,
R_k = offered load to demand pair k,
B_{jk} = blocking on path j of demand pair k.

(The dependence of these quantities on the hour has been suppressed for clarity.)

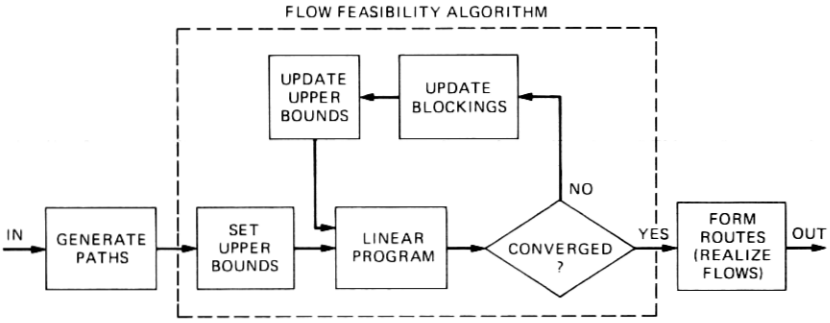


Fig. 10—Unified algorithm path formulation router detail.

Hence, the initial upper bound on flow is set, assuming that the entire offered load can be offered to any path independently of the load offered to any other path. Thus, the resulting flows can be infeasible since there might not be enough offered load to simultaneously achieve the desired flow on all paths for the same demand pair. For instance, suppose

$$\begin{aligned} B_{1k} &= 0.2, \\ B_{2k} &= 0.1, \\ B_{3k} &= 0.2, \\ R_k &= 10 \text{ erlangs}, \end{aligned}$$

Then

$$\begin{aligned} UPBD_{1k} &= 8 \text{ erlangs}, \\ UPBD_{2k} &= 9 \text{ erlangs}, \\ UPDB_{3k} &= 8 \text{ erlangs}. \end{aligned}$$

We assume that the required GOS is 0.005 so that the flow on all three paths should total 9.95 erlangs; this required flow is feasible since an overall blocking of $B_{1k}B_{2k}B_{3k} = 0.004$ is possible should all paths be used. Now suppose that the LP chooses for this demand pair the optimal flows

$$\begin{aligned} r_{1k} &= 8 \text{ erlangs}, \\ r_{2k} &= 1.95 \text{ erlangs}, \\ r_{3k} &= 0, \end{aligned}$$

where r_{ik} is now redefined as the carried flow on path i of demand pair k. The only way to realize the desired flow of 8 erlangs on path 1 is to offer path 1 the entire 10 erlangs. This means that 2 erlangs will overflow path 1. These 2 erlangs can then be offered to path 2, but can result in a maximum flow of 1.8 erlangs due to the blocking on path 2. Hence, the desired flows are infeasible. A method to compute new upper bounds to force these flows toward a more feasible solution will be discussed shortly; attention will now be focused on the structure of the LP used with the path formulation.

An LP to optimize path flows will solve the following problem:

minimize

$$\sum_{i=1}^{L} M_i a_i$$

subject to

$$\sum_{k=1}^{K} \sum_{j=1}^{J_k^h} P_{jk}^{ih} r_{jk}^h \leq a_i \qquad i = 1, 2, \cdots, L$$

$$h = 1, 2, \cdots, H$$

$$\sum_{j=1}^{J_k^h} r_{jk}^h = G_k^h \qquad \begin{array}{l} h = 1, 2, \cdots, H \\ k = 1, 2, \cdots, K \end{array}$$

$$r_{jk}^h \leq UPBD_{jk}^h \qquad \begin{array}{l} h = 1, 2, \cdots, H \\ k = 1, 2, \cdots, K \\ j = 1, 2, \cdots, J_k^h \end{array}$$

$$r_{jk}^h \geq 0, \qquad a_i \geq 0,$$

where we redefine

$P_{jk}^{ih} = 1$ if path j for demand pair k uses link i in hour h,
$= 0$, otherwise,
$r_{jk}^h =$ carried load on path j for demand pair k in hour h,
$J_k^h =$ number of paths for demand pair k in hour h,
$G_k^h =$ total carried load for demand pair k in hour h.

The total carried load for demand pair k in hour h is related to the total offered load for demand pair k in hour h, as follows. The minimum blocking that can be achieved on demand pair k is

$$E_k^h = \prod_{j=1}^{J_k^h} B_{jk}^h,$$

where $B_{jk}^h =$ blocking on path j for demand pair k in hour h. Let

$$\text{GOS} = \text{desired grade-of-service}$$

and the blocking on demand pair k in hour h will be

$$f_k^h = \max[E_k^h, \text{GOS}].$$

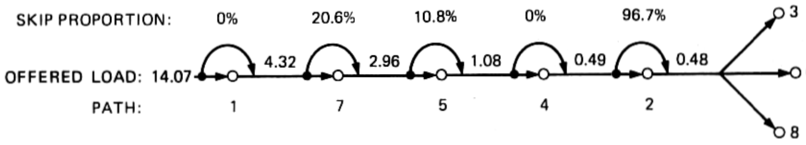
Then,

$$G_k^h = R_k^h[1 - f_k^h].$$

Thus, the total carried flow is determined by the GOS, unless E_k^h is greater than this desired GOS. If the GOS constraint cannot be met, all paths are required to be at their maximum flow to minimize the blocking. A blocking correction algorithm, similar to that used in the route formulation, is used in the engineering stage to correct those routes whose blockings are unacceptable.

Returning to Fig. 10, the next step in the flow feasibility algorithm is to update the link blockings in all hours based on the current link flow. This can be done by calculating the link size so that the maximum allowed blocking in any hour is not exceeded, and then calculating the blocking in all hours. After the blockings have been updated, the upper bounds need to be recalculated based on the current desired flows (determined by the LP), so as to obtain a more feasible solution.

The method used to recalculate the upper bounds is best illustrated by an example. The data in Fig. 11 show how a routing method, called



1 PATH NUMBER	2 PATH BLOCKING	3 LP CARRIED LOAD	4 LP OFFERED LOAD	5 REALIZED OFFERED LOAD	6 REALIZED CARRIED LOAD	7 UPPER BOUND	8 VIOLATION
1	0.307	9.75	14.07	14.07	9.75	9.75	0
7	0.603	1.36	3.43	3.43	1.36	1.72	0
5	0.287	1.88	2.64	2.64	1.88	2.11	0
4	0.453	1.06	1.94	1.08	0.59	0.59	0.47
2	0.239	0.012	0.016	0.016	0.012	0.37	0
3	0.309	0	0	0.48*	0	0.33	0
6	0.339	0	0	0.48*	0	0.32	0
8	0.488	0	0	0.48*	0	0.25	0

*OFFERED LOAD ASSIGNED FOR UPPER BOUND CALCULATION AS DESCRIBED IN THE TEXT.
EXCEPT FOR PATH NUMBER AND PATH BLOCKING, ALL ENTRIES ARE IN ERLANGS.

Fig. 11—Upper bound determination using a skip-one-path algorithm.

skip-one-path routing, can be used to set upper bounds which force more feasible flows, while still allowing the LP some flexibility in choosing new flow patterns. (The algorithm is called skip-one-path because traffic is allowed to skip a path where it is not needed.) Basically, this algorithm works by keeping track of the offered load available and using this load to realize the desired flows in a sequential manner. The data in Fig. 11 were the flows selected by an LP that used the initial upper bounds to route 14.07 erlangs of load on a particular demand pair in a particular hour.

The first step in the algorithm is to calculate the load which must be offered to each path to realize the flow selected by the LP. This offered load, given in the fourth column of Fig. 11, has been calculated from the carried load on each path selected by the LP (column 3) divided by one minus the blocking on the path (column 2). The next step is to sort the path offered loads from largest to smallest. This has been done for the data in Fig. 11; note that path number 7 follows path number 1 in terms of offered load. The path numbers used here refer to an internal ordering used by the algorithm.

Once the path ordering has been determined, the algorithm proceeds as follows. As the largest offered load desired by the LP is equal to the total offered load of 14.07 erlangs, all the load must be offered to path 1 as shown in the diagram in Fig. 11. Hence, none of the offered load is "skipped over" path 1. Applying the load in this way will realize the desired flow on the first path. Note that path 1, which carries the greatest flow, is at its upper limit (a common situation). With the given blocking of path 1, the overflow from path 1 is 4.32 erlangs.

Thus, the offered and carried loads desired on path 1 can be achieved, as shown in columns 5 and 6. Since the total demand load is available for path 1, and the blocking is assumed constant for this example, the upper bound on path 1 flow remains constant. The last column in Fig. 11 gives the violation, or amount by which the desired flow exceeds the new upper bound. In this case, the violation is zero.

Now consider path 7, which is next in order of offered load. The desired offered load to this path is 3.43 which is less than the overflow from path 1. The difference between these two loads, which is 4.32 − 3.43 = 0.89 (20.6 percent of 4.32), is skipped over path 7, and 3.43 erlangs is applied to path 7. This process of skipping can be accomplished by generating a random number before the call is offered to path 7. With probability 0.206, a call skips path 7 and is offered to the next path. A call that does not skip is offered to path 7.

Thus, the desired flow on path 7 can be realized. The upper bound on path 7 is calculated assuming the entire offered load (4.32 erlangs) could be offered to path 7. Note that this allows for more flow on path 7, if desirable, on the next iteration of the LP. The skip-one-path algorithm gives an actual offered load to path 7 of 3.43 erlangs with 2.07 erlangs overflow. The overflow is calculated as (0.603) (3.43) = 2.07. The total available load for any other path is now 2.07 + 0.89 = 2.96 erlangs.

Now consider path 5 which needs 2.64 erlangs of offered load. The amount of traffic to be skipped is 2.96 − 2.64 = 0.32, or 10.8 percent of the 2.96 erlangs available. The upper bound on the path 5 flow is based on 2.96 erlangs, which is the total available load at present that could be offered to path 5.

A different situation arises, however, when attempting to realize the desired offered load to path 4 of 1.94 erlangs. The total of the overflow from path 5 (0.76 erlangs) and the 0.32 erlangs skipped over path 5 is 1.08 erlangs which is the maximum load that can be offered to path 4. Hence, the LP has assigned more flow than can be realized. The maximum possible flow is 0.59; likewise, the upper bound is 0.59. Thus, there is a bound violation of 1.06 − 0.59 = 0.47 erlangs.

The process continues until the last path with a nonzero LP flow has been dealt with. At this point, all unused load, 0.48 erlangs in this example, is assumed to be available as offered load for all paths with a zero flow assigned by the LP. The upper bounds are then calculated in the same way as the initial upper bounds were set.

As mentioned earlier, the algorithm is called skip-one-path because traffic is allowed to skip a path where it is not needed. Actually, given that the amount of load to be skipped can be realized by generating random numbers, this algorithm yields a workable routing method to realize the desired LP flows, as will be discussed in Section 3.5.4.

Once the upper bounds have been calculated, the LP can be again executed to optimize the new problem. The sum of bound violations is available as a measure of flow feasibility. It is not necessary to begin the LP from "scratch" since the current routing patterns, with upper bounds updated to reflect the new flows, can be used as a starting basis.

3.5.2 Flow realization techniques

Once the path LP has converged, we must then realize the LP flows by forming the appropriate routes. The flow realization algorithm selects the routes. Three flow realization algorithms are discussed here and differ in their computational complexity and their flexibility in approximating the desired flows. Each algorithm treats the desired flows in each design hour independently, hence, the routing changes from hour to hour.

3.5.3 Routing algorithm-CGH

The CGH algorithm, named after Chung, Graham, and Hwang, who developed it, is composed of cyclic blocks. For example, suppose there are seven paths with desired flows r_i. One possible cyclic block realization of the seven r_i is

$$(1)\ (2\ 3\ 4)\ (5\ 6)\ (7).$$

The notation means that all the offered load to this route is first offered to path 1. The overflow from path 1 is then offered to a cyclic block composed of paths 2, 3, and 4. The term cyclic block means that a proportion β_i^k of the total load offered to the kth block is offered to cyclic permutation i, where cyclic permutation i is selected so that the ordering within the block is preserved but a different path appears first. In the cyclic block under consideration, a proportion β_1^2 of the input traffic will be offered to the paths in the order (2, 3, 4) and proportion β_2^2 to (3, 4, 2), etc. Offering traffic in this manner may be accomplished by generating a random number when a call is offered to the cyclic block. Note that all calls see the same blocking probability within the cyclic block since all paths are searched.

The realization algorithm must define the contents of each cyclic block and calculate the proportions β_i^k associated with the kth cyclic block. The basic steps to accomplish this are as follows, and the subsequent example should make the steps clear.

In the interest of brevity, notation dealing with demand pairs and design hours has been suppressed. Let

r_i = desired flow on path i,
B_i = blocking associated with path i,
$Q_i = 1 - \mathrm{B}_i$ = connectivity of path i,

$\sigma_i = \dfrac{r_i}{Q_i}$ = desired offered load to path i,

$\delta_i = B_i\sigma_i$ = desired overflow load from path i,

J = total number of paths.

(*i*) Calculate σ_i and δ_i. Sort and relabel the σ_i, if necessary, so that

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \cdots \geq \sigma_J.$$

(*ii*) The first path in the cyclic block to be formed is the as yet unused path i with the largest σ_i.

(*iii*) Insert an as yet unused path i with largest remaining σ_i after a path j with $\sigma_i > \delta_j$, if such a path j exists. Repeat this step until no such j exists.

(*iv*) The current (kth) cyclic block ends with the last path inserted. If there is but one path in the block, set its coefficient to 1.0 and go to (*v*). If there is more than one path in the block, let

$$\alpha_l^k = \sigma_{m(l)} - \delta_j,$$

where $m(l)$ refers to the path in the lth position in the kth block, and j refers to the path preceding it in the cyclic ordering of the block. Note that the algorithm guarantees that all α_l^k are positive. Then calculate

$$\beta_i^k = \frac{\alpha_i^k}{\sum_{i=1}^{L} \alpha_i^k},$$

which is the cyclic coefficient associated with the ith path in the kth block, assuming there are L paths in the block.

(*v*) If there are remaining $r_i > 0$, return to (*ii*).

(*vi*) Add single path cyclic blocks at the end of the route, if necessary, until the GOS constraint is satisfied.

Table III shows an example of the algorithm and the resulting

Table III—The CGH routing example

Path Number	Path Blocking	LP Carried Load	LP Offered Load	LP Overflow Load	Realized Carried Load	Error
	(B_i)	(r_i)	(σ_i)	(δ_i)		
1	0.307	9.75	14.07	4.32	9.75	0
7	0.603	1.36	3.43	2.06	1.26	0.10
5	0.287	1.88	2.64	0.76	1.74	0.14
4	0.453	1.06	1.94	0.88	0.98	0.08
2	0.239	0.012	0.016	0.0037	0.26	0.25
3	0.309	0	0	0	0.06	0.06
6	0.339	0	0	0	0	0
8	0.488	0	0	0	0	0
					Total:	0.63

Route: (1) (7, 5, 4) (2) (3)
Coefficients: (100%) (59.2%, 13.4%, 27.4%) (100%) (100%)
Except for path number and path blocking, all entries are in erlangs.

routing. The data for this example is identical to that used for the example in Fig. 11. The path order in Table III has already been sorted on offered load (σ_i). Path 1 becomes the first path in the first cyclic block since σ_1 is the largest; it is also the only path in the first cyclic block since no other σ_i is larger than δ_1.

Path 7 begins the next cyclic block, since σ_7 is the largest remaining offered load. Path 5 follows path 7 in the second cyclic block since σ_5 (2.64) is greater than δ_7 (2.06). Likewise, path 4 follows path 5. The second block ends with path 4, since no other unused path has an offered load greater than δ_4.

The coefficients of the second cyclic block are computed as follows:

$$\begin{aligned} \alpha_1^2 &= \sigma_7 - \delta_4 = 3.43 - 0.88 = 2.55 \\ \alpha_2^2 &= \sigma_5 - \delta_7 = 2.64 - 2.06 = 0.58 \\ \alpha_3^2 &= \sigma_4 - \delta_5 = 1.94 - 0.76 = \underline{1.18} \\ \text{Total} & \qquad\qquad\qquad\qquad\quad = 4.31 \end{aligned}$$

Then,

$$\beta_1^2 = \frac{2.55}{4.31} = 59.2 \text{ percent}$$
$$\beta_2^2 = \frac{0.58}{4.31} = 13.4 \text{ percent}$$
$$\beta_3^2 = \frac{1.18}{4.31} = 27.4 \text{ percent}$$

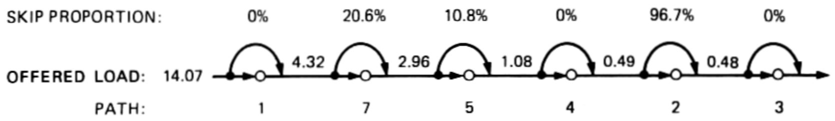
These coefficients β_i^k for the four blocks are shown below the route shown in Table III in order of starting path; thus, 59.2 percent of the traffic offered to the second block starts with path 7.

Path 2 forms a one-member cyclic block since it is the only path left with a positive σ. Note that path 3 was included to decrease the blocking from 0.0057 to 0.0017, thus, meeting the GOS objective of 0.005. The total path flow error (absolute difference between desired flow and realized flow) is shown to be 0.63 erlangs.

3.5.4 Skip-one-path algorithm

The skip-one-path algorithm can be used to realize path flows, as well as to calculate upper bounds. An example of skip-one-path routing is shown in Fig. 12. A No. 4 ESS routing data block could be modified to do skip-one-path routing by generating a random number before a call is offered to the next path. With a predetermined probability, a call would skip over the path without being offered to it and proceed to the next path in the routing sequence.

Once again, the first step is to sort the paths by offered load. The algorithm used to calculate the amount of offered traffic to skip the next path was discussed previously. Note that path 3 has been added



PATH NUMBER	PATH BLOCKING	LP CARRIED LOAD	LP OFFERED LOAD	REALIZED OFFERED LOAD	REALIZED CARRIED LOAD	|ERROR|
1	0.307	9.75	14.07	14.07	9.75	0
7	0.603	1.36	3.43	3.43	1.36	0
5	0.287	1.88	2.64	2.64	1.88	0
4	0.453	1.06	1.94	1.08	0.59	0.47
2	0.239	0.012	0.016	0.016	0.012	0
3	0.309	0	0	0.48	0.33	0.33
6	0.339	0	0	0	0	0
8	0.488	0	0	0	0	0
					TOTAL:	0.801

EXCEPT FOR PATH NUMBER AND PATH BLOCKINGS, ALL ENTRIES ARE IN ERLANGS

Fig. 12—Skip-one-path routing example.

to meet a GOS objective of 0.005. Also, Fig. 12 shows a path flow error of 0.80, which is greater than the 0.63 path flow error given by the CGH algorithm.

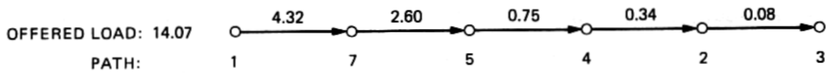
3.5.5 Sequential routing algorithm

A very simple method to realize desired path flows is termed sequential routing. This scheme simply sorts the desired flows on offered load (as do the other methods) and lets the first path overflow to the second path which overflows to the third path, and so on. Thus, traffic is routed sequentially from path to path with no probabilistic methods being used to get the realized flows closer to the desired flows. The reason that sequential routing works well is that most flow is carried on the first one or two paths, which are loaded to their upper bound, and errors in meeting flow on later paths are not significant.

Figure 13 shows a sequential routing example. The given blockings and desired flows are identical to those used in the other routing examples. Note that in this particular example, sequential routing has the highest error in flows of all the three routings studied. In general, sequential routing has the least flexibility of the three realization methods discussed here. We consider the effect of this flow inaccuracy on network cost.

3.6 Path formulation results and conclusions

Network designs were obtained for the CGH algorithm and the sequential routing algorithms using a 30-node network model. The results in Table IV show that the CGH algorithm is more accurate than



PATH NUMBER	PATH BLOCKING	LP CARRIED LOAD	LP OFFERED LOAD	REALIZED OFFERED LOAD	REALIZED CARRIED LOAD	|ERROR|
1	0.307	9.75	14.07	14.07	9.75	0
7	0.603	1.36	3.43	4.32	1.72	0.35
5	0.287	1.88	2.63	2.60	1.86	0.02
4	0.453	1.06	1.94	0.75	0.41	0.65
2	0.239	0.012	0.016	0.34	0.26	0.25
3	0.309	0	0	0.08	0.06	0.05
6	0.339	0	0	0	0	0
8	0.488	0	0	0	0	0
					TOTAL:	1.32

EXCEPT FOR PATH NUMBER AND BLOCKING, ALL ENTRIES ARE IN ERLANGS

Fig. 13—Sequential routing example.

the sequential algorithm, but the difference in final network cost was only about 0.5 percent. Additionally, these routing methods added between 1 and 2 percentage points to the savings achieved with the route formulation. While these results have been illustrated only for small network models, they have recently been confirmed using a full-scale, 215-node, intercity network model.

Hence, among path formulation routing alternatives, the very simple sequential routing technique achieves network design savings almost identical to those of much more complicated schemes. A routing method, such as CGH has additional costs not quantified in this study. For instance, a switching system with CGH routing would have to store traffic allocation proportions and markers to indicate where the cyclic blocks begin and end, along with the ordered list of paths. Sequential routing, on the other hand, needs only the ordered list of paths. Also, applying traffic allocation techniques such as those needed by CGH routing would require real time to generate and process the appropriate random numbers. Sequential routing needs no such traffic allocation and, hence, has a real-time advantage.

Table IV—Network designs for 30-node network (based on 16 hours)

Network Routing	Cost	Savings (%)
Hierarchical	$137,874,300	
Two-link (route formulation)	117,830,000	14.5
Sequential	115,534,300	16.2
CGH	114,849,300	16.7

IV. OPTIMIZATION METHODS AND RESULTS

4.1 Heuristic optimization method (HOM)

As mentioned in Section 3.2, an HOM was developed to solve the LP problems of the UA route formulation. This heuristic was revised to solve the LP problems of the path formulation (Section 3.4). For the sake of brevity, this section describes only the latter version. We discuss the three basic ideas underlying the HOM and provide a brief overview.

4.1.1 Rerouting of traffic

The first concept concerns the rerouting of traffic. A reroute is a reassignment of flow of a particular point-to-point pair from one path to another in a single design hour. Given an initial assignment of path flows for each point-to-point pair, the HOM progresses to its final solution by a sequence of reroutes. Thus, each iteration of the heuristic affects the flow on a few links and in only one design hour.

4.1.2 Marginal costs

Next, we discuss a concept which allows us to evaluate the potential cost savings of any reroute. The marginal link cost is an estimate for the rate of change of the total network cost function relative to the change in flow on a link during a particular design hour. The HOM uses an UPCOST and a DOWNCOST indicating the predicted cost change if we increase or decrease the link flow during a particular hour. We maintain marginal costs for every link during every design hour.

The rules for determining the marginal link costs of a link are simple. For a particular link we examine the flow during each design hour. If the peak flow on the link occurs in only one hour, then increasing or decreasing the flow in that hour will increase or decrease the capacity of the link. We then set the UPCOST and DOWNCOST in the peak hour equal to the metric of the link. If the peak flow occurs in more than one hour, then increasing the flow in one of the peak hours will increase the link capacity, while decreasing the flow in one of the peak hours will leave the capacity unchanged. We then set the UPCOST in all peak hours equal to the link metric and set the DOWNCOST in all peak hours equal to zero. In all design hours where the link flow is below the peak flow, we set the UPCOST and DOWNCOST equal to zero since increasing or decreasing the flow does not affect the link capacity.

Once the marginal link costs are computed, we can determine the marginal cost of diverting flow from one path to another in the following way. We first sum the UPCOSTs of the path that will gain flow, and then sum the DOWNCOSTs of the path that will lose flow.

Subtracting the latter sum from the former sum yields the marginal cost of the reroute. If this cost is negative, then the reroute is profitable.

Once we decide to perform a particular reroute whose marginal cost indicates that it is profitable, we then determine the amount of flow to divert. The rule for finding this quantity is to continue rerouting until the marginal cost of the reroute changes.

Figures 14 and 15 sketch an example of how the marginal link costs are determined and how they are modified when a rerouting of traffic occurs. Figure 14 shows two paths between nodes A and D in the first of two design hours. Each path has two links. The metrics for links AB, BD, AC, and CD are 10, 10, 60, and 60, respectively. Path 1 is initially assigned 20 erlangs of flow while path 2 is assigned none. The upper bounds on the path flows are 20 and 15 erlangs, respectively.

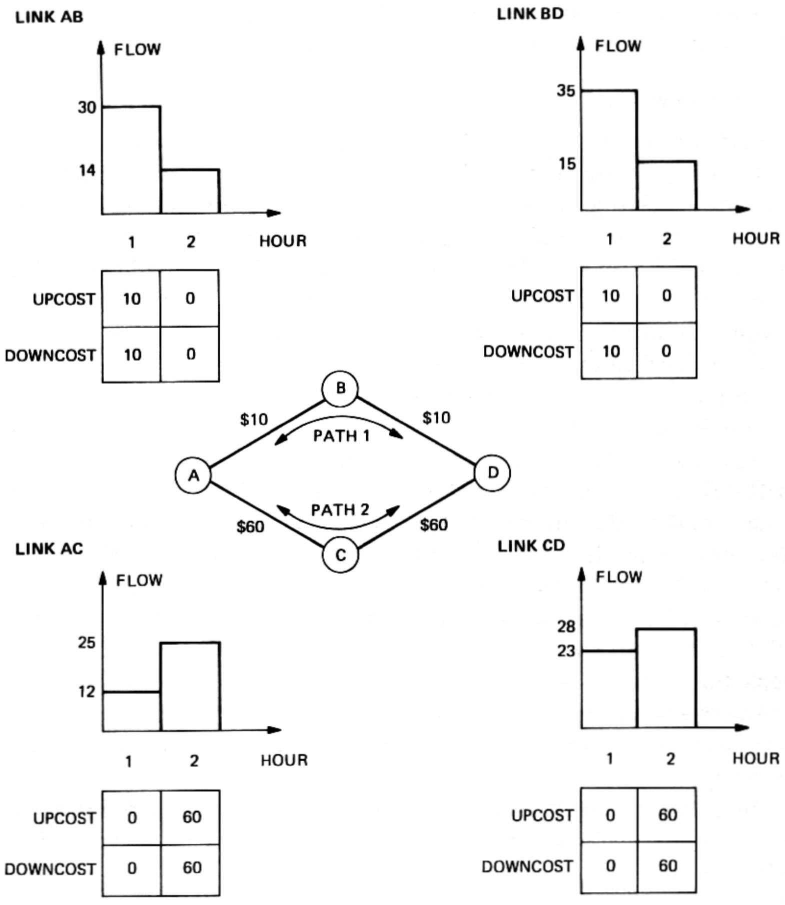


Fig. 14—Link flows and marginal link costs in example.

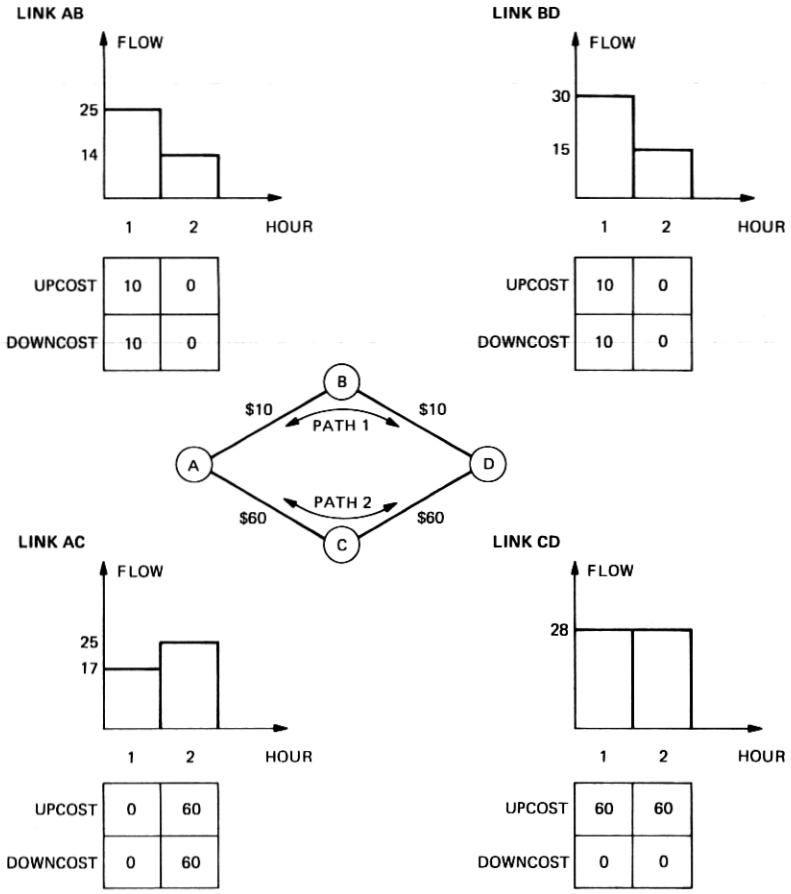


Fig. 15—Updated link flows and marginal link costs in example.

Figure 14 also shows the initial flow and marginal costs for each of the links in each of two design hours. For example, link AB carries 30 erlangs in hour 1 and only 14 erlangs in hour 2. Since it has a unique peak in hour 1, the UPCOST and DOWNCOST of link AB in hour 1 are set equal to 10, the metric of link AB. In hour 2, they are set equal to zero.

We can now use the marginal link costs to determine the marginal cost for rerouting traffic from path 1 to path 2. Summing the UPCOSTs of links AC and CD in hour 1, and subtracting from it the sum of the DOWNCOSTs of links AB and BD in hour 1 yields

$$(0 + 0) - (10 + 10) = -20,$$

the marginal cost of diverting from path 1 to path 2 in hour 1. Therefore, the reroute is a profitable one.

We next determine the amount of flow to divert. Now the marginal profit of the reroute will hold until one of the marginal link costs in the above calculation changes. We then reroute as much flow as possible until either the DOWNCOST of link AB, the DOWNCOST of link BD, the UPCOST of link AC, or the UPCOST of link CD changes in hour 1. Figure 15 describes the effect of rerouting 5 erlangs of flow in hour 1. Link CD has gained 5 erlangs of flow and now has 2 peak hours. The UPCOST in each hour is now equal to the link metric, while the DOWNCOST in each hour is zero. Since the UPCOST of link CD has changed from zero to sixty, 5 erlangs is the total amount of flow that we reroute. If after the marginal reroute cost is reevaluated we find that the reroute is still profitable, we continue to divert flow until the marginal reroute cost changes again. We continue in this manner until either the reroute is no longer profitable, or there is no more flow assigned to path 1, or the flow on path 2 reaches its upper bound. We then search for another profitable reroute.

4.1.3 Candidate list

The last concept for the HOM concerns the method for deciding how many candidate reroutes to evaluate before actually performing a particular reroute. In selecting a reroute pair, there is a tradeoff between the quality of the reroute found and the amount of time spent searching for it. Although we would like to find very profitable reroutes, the HOM should also be computationally efficient. The HOM uses a candidate list to find the next reroute to perform. This concept works in the following way. The first M point-to-point pairs are searched for profitable reroutes. The K most profitable reroutes are put into a candidate list and the most profitable reroute in the list is selected and performed. Once this particular reroute is no longer profitable, the remaining members of the list are reevaluated and the most profitable reroute is selected and performed. This process continues until there are no more profitable reroutes left in the list. The next M point-to-point pairs are then searched for profitable reroutes, a new list is generated, and the reroutes in the list are performed until they are no longer profitable. The HOM continues in this manner. Whenever the last point-to-point pair in the set of all point-to-point pairs is encountered, the next point-to-point pair to be considered is the first point-to-point pair in the set. The heuristic then "wraps around" the set of all point-to-point pairs. The HOM finally terminates when there are no profitable reroutes among all point-to-point pairs.

4.1.4 Overview

The three concepts we have described in this section (the rerouting of traffic, the marginal costs, and the candidate list) are used together in the HOM. After an initial feasible solution is selected, the marginal link costs are determined. A group of point-to-point pairs is searched and a list of the most profitable reroutes is formed. Each reroute is performed until it is no longer profitable. When there are no more profitable reroutes in the list, a new group of point-to-point pairs is searched and a new list is formed. The heuristic continues in this manner until there are no profitable reroutes. The next section describes typical examples of the computational results that have been obtained.

4.2 Computational results

We compared the HOM with MPSX/370 using an LP problem that was generated by the UA. The problem was derived from the 28-node network where the average number of paths per point-to-point pair was 9.4. The corresponding LP had 1402 rows, 7630 columns, and a density of 0.17 percent. We used as a reference point a nonoptimal solution obtained by MPSX/370 after 904 CPU seconds. The heuristic progressed very rapidly until it was within 0.2 percent of the MPSX/370 solution. It then terminated after only 2 CPU seconds. In contrast, MPSX/370 required 860 CPU seconds to produce a solution of similar quality. We see that the HOM can produce a near-optimal solution much more quickly than MPSX/370. Also, the UA contains many approximations so that an optimal solution to the LP is not necessary. In fact, there is little penalty in the network cost if the HOM is used.

4.3 Run times for large networks

To design a 200-node network with 6 design hours, the UA will require about 20 million bytes of memory, much of which is needed to solve the linear programs.

Tests with a 190-node intercity network model for 6 design hours indicate that the UA will require less than 4 hours of CPU time to design a large network. Half of this time will be spent by the HOM. With expected advances in hardware, the total time may become considerably smaller.

V. POTENTIAL BELL SYSTEM APPLICATIONS

The implementation of DNHR requires the following developments:

(*i*) Network design, servicing, and electronic switching system software.

(*ii*) Collection of point-to-point data. The network would be designed and administered using a point-to-point blocking criterion and the blocking must be measured to administer the network properly.

(*iii*) Mechanization of routing administration function. The proliferation of routing updates would necessitate a mechanized routing administration system.

(*iv*) Design along a network boundary separating the centralized intercity network from the decentralized metropolitan networks. This is to achieve efficient network designs, both in cost and in computing time.

(*v*) Modification of network operation support systems such as the network management systems.

(*vi*) Modifications of switch planning tools and methods. These must be modified to reflect DNHR design in order to model the network properly.

The expected benefits of dynamic routing are attractive. However, the fundamental nature of the changes raises service, cost, and feasibility issues that might substantially reduce the projected benefits.

Several studies are underway to assess fundamental issues, such as network management, switching and signaling loads, large scale optimization, and transmission performance.

VI. ACKNOWLEDGMENTS

R. T. Wong originally conceived and implemented the route formulation heuristic method and has made substantial contributions to many other aspects of this work. We also acknowledge the contribution of A. H. Westreich, who is responsible for the work on progressive routing. F. R. K. Chung, R. L. Graham, and F. K. Hwang were instrumental in developing the path formulation of the UA and developed the CGH routing algorithm. L. T. Nguyen has made substantial contributions to many of the algorithms discussed here. He also developed fast traffic algorithms which were used in the design of the very large networks reported here.

REFERENCES

1. F. R. K. Chung, R. L. Graham, and F. K. Hwang, Efficient Realization Techniques for Network Flow Patterns, B.S.T.J., this issue.
2. G. R. Ash, A. H. Kafker, and K. R. Krishnan, "Servicing and Real-Time Control of Networks with Dynamic Routing," B.S.T.J., this issue.
3. C. J. Truitt, "Traffic Engineering Techniques for Determining Trunk Requirements in Alternate Routed Networks," B.S.T.J., *33*, No. 2 (March 1954), pp. 277–302.
4. B. Yaged, Jr., "Long Range Planning for Communications Networks," Polytechnic Institute of Brooklyn Ph.D Thesis, 1971.
5. J. E. Knepley, "Minimum Cost Design for Circuit Switched Networks," Technical Note Numbers 36-73, Defense Communications Engineering Center, System Engineering Facility, Reston, Va., July, 1973.
6. W. Feller, *An Introduction to Probability Theory and Its Applications*, Third Edition, New York: John Wiley, 1968.