Report Number: WUCS-91-55

1992-01-10

Designing Communication Networks with Fixed or Nonblocking Traffic Requirements

Authors: J. Andrew Fingerhut

A general framework for specifying communication network design problems is given. We analyze the computational complexity of several specific problems within this framework. For fixed multirate traffic requirements, we prove that a particular network analysis problem is NP-complete, although several related network design problems are either efficiently solvable or have good approximation algorithms. For the case when we wish the network to operate without blocking any connection requests, we give efficient algorithms for dimensioning the link capacities of the network.

Designing Communication Networks with Fixed or Nonblocking Traffic Requirements

J. Andrew Fingerhut

WUCS-91-55

January 10, 1992

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

A general framework for specifying communication network design problems is given. We analyze the computational complexity of several specific problems within this framework. For fixed multirate traffic requirements, we prove that a particular network analysis problem is NP-complete, although several related network design problems are either efficiently solvable or have good approximation algorithms. For the case when we wish the network to operate without blocking any connection requests, we give efficient algorithms for dimensioning the link capacities of the network.

This work is supported by the National Science Foundation, Bell Communications Research, Bell Northern Research, Digital Equipment Corporation, Italtel SIT, NEC, NTT, and SynOptics.

1. Introduction

Much work has been done on the computational problem of designing low-cost communication networks (see [GN89, GTD+89, GW90, GK90, KKG91, AKR91] and references therein). The general problem is: given a collection of nodes, a set of node pairs which are allowed to be connected by links, costs for installing links of various capacities, traffic requirements between nodes, and possibly other constraints, choose which capacity to install on each link such that the traffic requirements are satisfied, the "other" constraints are satisfied, and the total cost is minimized.

Gersht and Weihmayer [GW90] include as part of their "other" constraints a restriction that the average end-to-end packet delay is below a given threshold. Gavish and Neuman [GN89] do not restrict the end-to-end packet delay below a threshold, but they include it as part of the cost of the network. Their approaches concentrate on mathematical programming techniques for formulating and solving the problems. We are not concerned with modeling or restricting packet delay in this work.

Agrawal, Klein, and Ravi [AKR91] give an algorithm for a network design problem with fixed point-to-point requirements and link costs equal to a startup cost, independent of capacity. Their algorithm runs in polynomial time, and is guaranteed to produce a solution no worse than twice optimal. This result is extended to allow multipoint requests as well in Section 4.3.

Gavish et al. [GTD+89] give a heuristic for solving a network design problem with fixed point-to-point requirements and link costs equal to an arbitrary function. It is based on the simplex method of linear programming. This algorithm is not guaranteed to find good solutions, but appears to work well in practice.

Kershenbaum et al. [KKG91] give a heuristic for solving a network design problem with fixed point-to-point requirements and link costs proportional to the capacity, where capacity is only available in increments which are large relative to the individual traffic requirements. It is remarkable because of its fast $O(n^2)$ running time, where n is the number of nodes in the network. It is not guaranteed to find good solutions.

Certain highly structured networks, such as Clos and Cantor networks, are nonblocking when used for circuit switching (if a route uses a link, it uses the entire capacity of the link). Melen and Turner [MT89] consider using these networks for multirate connection requests, where each request can use a fraction of a link's capacity in the range $[b, B]$. They assign every external link or "terminal" a bandwidth of 1 and every internal link a bandwidth of $1/\beta$, where $\beta \leq 1$, and then proceed to find sufficient conditions for which the network is nonblocking. These conditions consist of inequalities involving parameters of the networks and β. This is essentially dimensioning the internal links large enough so that the network is nonblocking. This idea is generalized in Section 5 to networks with arbitrary topology, terminals with arbitrary bandwidth (called termination capacity there), and a problem in which the objective is to determine each link capacity independently.

In this work, we place emphasis on network design problems which are appropriate for designing a campus-wide or metropolitan area network (MAN). These problems are formulated to focus on the central computational issues. We then use the tools of computational

complexity in order to discover which constraints cause the problems to be difficult to solve. Our desired goal is to find polynomial time algorithms which either find optimum solutions or solutions guaranteed to be close to optimum. Wherever possible, we allow multipoint traffic requirements as well as point-to-point.

Section 2 defines a general class of problems, and section 3 defines the restricted problems examined in this work. Section 4 discusses results on analyzing and designing networks with fixed traffic requirements, and section 5 examines networks which must be nonblocking. Ideas for future work are discussed in section 6.

2. The real-world design problem

Several structures arise when specifying an instance of the network design problem and when discussing solutions. A *physical graph* describes physical constraints for where equipment may be placed. *Equipment descriptions* specify the types and properties of physical objects which may be used in constructing the network. A *logical graph* describes a way of connecting equipment together to form the network, and an *embedding* specifies where to place equipment. *Traffic requirements* tell us how we expect to use the network. More detailed descriptions of these structures are contained in the following subsections.

2.1. Physical graphs

A *physical graph* represents the constraints placed on the location of equipment by the location (buildings, walls, etc.) in which the network will be installed. It may certainly be possible to alter such restrictions (tear down buildings, put up walls, etc.), but for our purposes, any such changes are considered either impossible or too costly.

A physical graph is an undirected multigraph $PHYS = (V, E)$. Each vertex $v \in V$ represents a *place* where switching equipment could be installed (e.g. a communication closet). Each place has several parameters which are important for the design problem:

- Volume available.
- Physical conditions (e.g., humidity, temperature).

There may be other parameters which have been overlooked here.

Each edge $e \in E$ represents a *link path* where one or more transmission links may be placed, e.g., a duct or chase in a building, a tunnel between buildings. Each link path has several parameters:

- Length
- Cross-sectional area available.
- Physical conditions (e.g., electromagnetic interference).
- Cost for laying cable on this link path.

An example physical graph is given in Figure 1. Rectangles represent places, and "fat lines" represent link paths. The items have been drawn with different sizes to suggest their respective volumes, cross-sectional areas, and lengths.

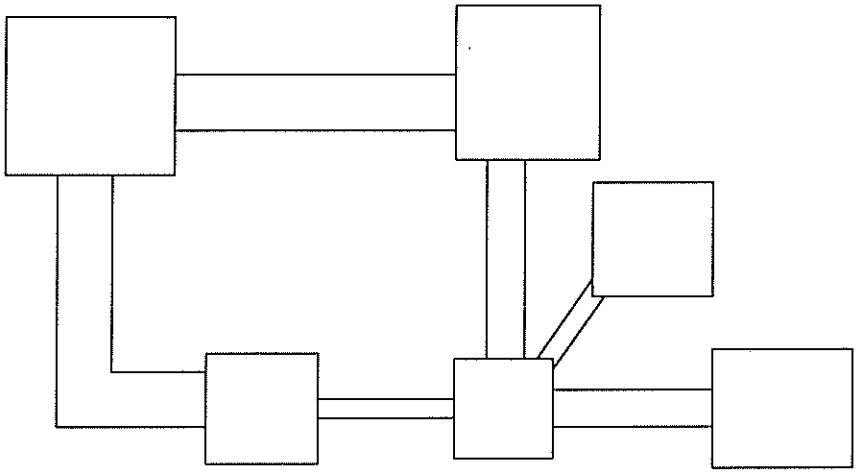


Figure 1: An example physical graph

2.2. Equipment descriptions

When designing a network, one must have in mind what sort of equipment is available for use, including its cost and performance characteristics. Here we assume that the available equipment fits into one of the following two classes.

A *switch* is a device which can have several transmission links attached to it. Information sent to the switch is routed through internal paths to an appropriate output link, and is then transmitted to some further destination. For a more detailed description of fast packet switching, see [Hui90, dP91].

There may be several types of switches available for use. Each type is parameterized by:

- Volume occupied.
- Total switching capacity.
- Cost, possibly including a startup cost and a continuing maintenance cost.
- Number and type of links which may be terminated at the switch.
- Need for a controlled environment.

A *link* is some type of cable (e.g., twisted pair wire, coaxial or fiber optic cable) which can transmit information. Each type is parameterized by:

- Cross-sectional area occupied.
- Maximum length over which a signal may be effectively carried.
- Bandwidth measured in bits/sec.
- Cost per unit length.
- Termination cost. This is the cost of connecting a cable to a switch, and may depend on the type of both.
- Need for controlled environment.

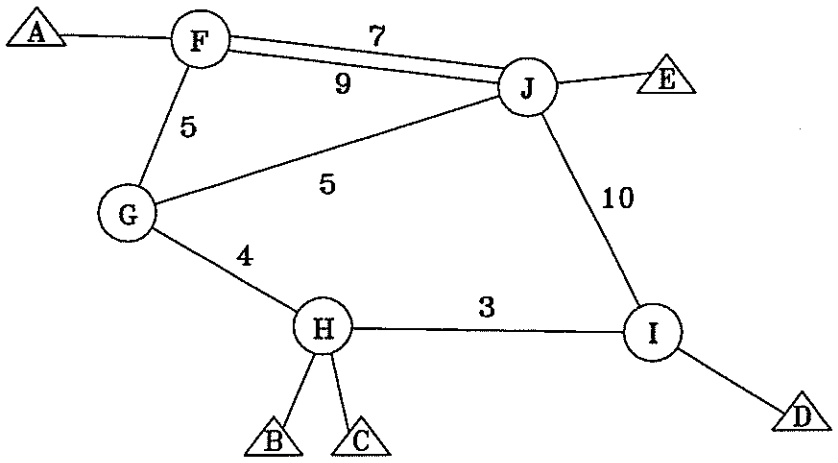


Figure 2: An example logical graph

The purpose of the network is to connect *terminals* together. Terminals can be sources and destinations of messages in the network. They are characterized by certain traffic requirements, which are explained in section 2.5. It is assumed here that all terminal locations are fixed.

2.3. Logical graphs

A *logical graph* represents a particular way of connecting switches, links, and terminals together. It does not specify anything about where such items are to be placed.

A logical graph is an undirected multigraph $LOG = (V, E)$. (This assumes that either the links are bidirectional, or that they are unidirectional and always placed in equal capacity, oppositely directed pairs.) The set of vertices V is partitioned into two sets, *terminals*(LOG) and *switches*(LOG). Each $v \in$ *terminals* represents a terminal, and each $v \in$ *switches* represents a switch with one of the given types. Each $e \in E$ represents a link with a given type. The characteristics of a link which are considered most important in this work are capacity and cost.

To be *feasible*, a logical graph must satisfy constraints given by the equipment descriptions, e.g., the number and type of links which may be terminated at a particular switch cannot be violated.

An example logical graph is given in Figure 2. Ovals represent switches, triangles represent terminals, lines represent links, and numbers represent the capacities of the links.

2.4. Embeddings

An *embedding* is a mapping from a logical graph LOG to a physical graph $PHYS$. It specifies where each piece of equipment is located. Vertices in LOG (switches and terminals) are

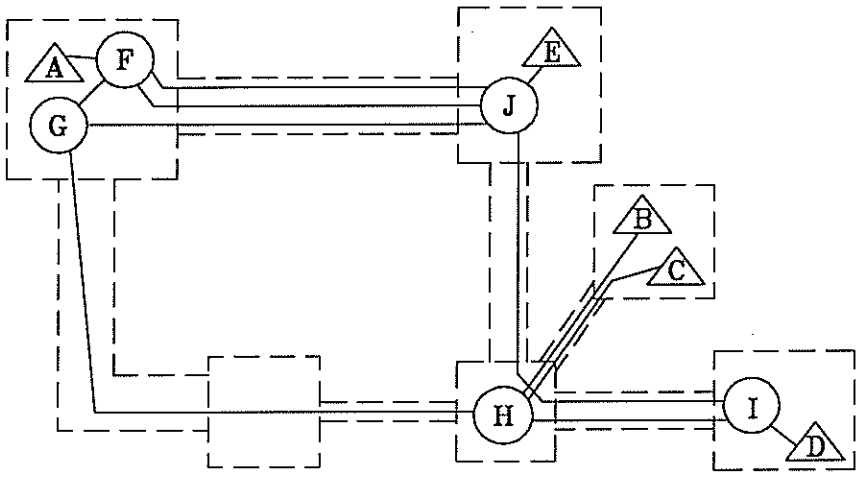


Figure 3: An example embedding

mapped to vertices in *PHYS* (places). Edges in *LOG* (links) are mapped to *paths* in *PHYS* (sequences of link paths).

An embedding is *feasible* if it satisfies constraints such as volume available in places, cross-sectional area available in link paths, and environment constraints. Taken together, a physical graph, equipment descriptions, a logical graph, and an embedding determine the cost of the network.

An example embedding is given in Figure 3. It is the graphs of Figure 1 and Figure 2 superimposed so as to suggest the mapping from the logical graph to the physical graph. The physical graph is drawn with dashed lines.

2.5. Traffic requirements

A network's purpose is to satisfy message traffic between terminals. An exact mathematical model of network traffic is difficult to specify and use, so simplified models are used. Examples include fixed requirements, nonblocking requirements, and stochastic requirements [GK90].

The following types of requirements are examined in this report: fixed point-to-point connection requests, fixed multipoint requests, and nonblocking point-to-point requests. All connections are specified with a peak rate; when the connection is active, the network reserves enough resources to satisfy the peak rate.

2.5.1. Fixed connection requests. Fixed connection requests describe an unchanging pattern of message traffic among terminals. They are not suitable in all cases, but they can be a good approximation to the real traffic, and the design problems and solutions which result can lend insight into other problems.

Fixed connection requests for a network are specified by a multiset *REQ* of individual requests. *REQ* will be referred to as a set even though it is a multiset. Individual requests q are specified by a pair (*req-terms*, *req-rate*), where *req-terms*(q) is a set of at least two terminals in the network, and *req-rate*(q) is a nonnegative integer specifying the rate of traffic for the request.

If a single connection request has exactly two terminals in its *req-terms* set, then it is called point-to-point, otherwise it is called multipoint. If every request in a set *REQ* is point-to-point, then the set is called point-to-point, otherwise it is called multipoint. If every request in a set *REQ* has the same value of *req-rate*, then the set is called single rate, otherwise it is called multirate.

A *route* r is a pair (*route-links*, *route-rate*). *route-links*(r) is a set of links in *LOG* which forms a tree whose leaves are terminals of *LOG*. *route-rate*(r) is the bandwidth which is used on each link of the route. A route r *realizes a request* q if the leaves of r are exactly the set of terminals *req-terms*(q) and *route-rate*(r) = *req-rate*(q). Note that for two terminals the route is simply a path. A *state* is a (multi)set of routes. A state s *realizes a set of requests REQ* if there is a one-to-one correspondence between each request q_i and each route r_i such that r_i realizes q_i.

$\mathcal{N}$ is used to denote the set of nonnegative integers. Given a logical graph *LOG* with link capacities $cap : E \to \mathcal{N}$ and a state s, we define the *usage* and *available capacity* of a link e in state s to be

$$\begin{aligned} usage(s,e) &= \sum_{r_i \in s,\, r_i \text{uses link } e} route\text{-}rate(r_i) \\ avail(s,e) &= cap(e) - usage(s,e) \end{aligned}$$

A state s is *compatible with logical graph LOG* if

$$(\forall e \in E)\,(usage(s,e) \leq cap(e))$$

In other words, every link is used in connections that have a total rate which is at most the link's capacity. We say that a logical graph *LOG* with given link capacities is *able to route a set of fixed requests REQ* if there exists a state s that realizes the requests *REQ* and is compatible with *LOG*. More briefly, we say *LOG is routable for REQ*.

For example, the set of fixed requests $REQ = \{(\{A,B\},3),(\{C,D,E\},1)\}$ can be realized by the state depicted in Figure 4. The state is superimposed upon the logical graph of Figure 2. The links are labeled with two numbers, first capacity and then usage. Note that the link $\{G,H\}$ has $usage(s,\{G,H\}) = 4 = cap(G,H)$ and $avail(s,\{G,H\}) = 0$. We call a link *saturated* if $avail(s,e) = 0$. The link $\{H,I\}$ has $usage(s,\{H,I\}) = 1$ and $avail(s,\{H,I\}) = 2$.

2.5.2. Nonblocking connection requests. In the nonblocking case, individual connection requests are defined exactly as in the fixed case above. However, instead of knowing the exact set of requests in advance, we only know some restrictions on the allowable request sets (those defined below as compatible). The network must be able to handle an arbitrary

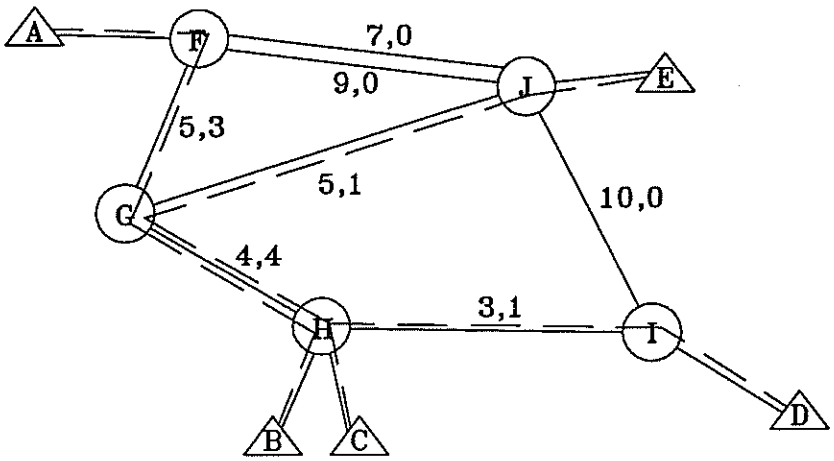


Figure 4: An example state

sequence of additions to and removals from the set of requests, as long as that set remains compatible.

Suppose we are given a logical graph LOG. For each terminal t, we specify a value $\beta(t)$ called the *termination capacity* of t. This specifies the maximum total rate of all connections in which t may be involved simultaneously. For example, if $\beta(t) = 5$, then t may be simultaneously involved in connections with rates 1, 2, and 2, but it could not be involved in any more connections until an existing connection is removed. In order to treat terminals and switches uniformly, it is convenient to define $\beta(v) = 0$ for all $v \in switches(V)$.

Given a logical graph LOG with termination capacities $\beta : V \to \mathcal{N}$ and a (multi)set of connection requests REQ, we define the *usage* and *available capacity* of a vertex v under requests REQ to be

$$\begin{aligned} usage(v, REQ) &= \sum_{q_i \in REQ,\, v \in req\text{-}terms(q_i)} req\text{-}rate(q_i) \\ avail(v, REQ) &= \beta(v) - usage(v, REQ) \end{aligned}$$

A request set REQ is *compatible with logical graph* LOG if

$$(\forall v \in V)\,(usage(v, REQ) \leq \beta(v))$$

That is, no terminal is involved in more requests than its termination capacity will allow.

Why introduce the notion of a termination capacity? There must be some way to restrict the number of connection requests that are allowed simultaneously. Later we will want to have algorithms to design link capacities given other information, so this restriction should not refer to link capacities. Termination capacity also models real world constraints well. A "terminal" here could be a terminal that has an interface to the network with some maximum bandwidth. It could also be a collection of terminals which access the network through a single multiplexor. The termination capacity models the bandwidth of the "terminal's" interface to the network. It's use was inspired by the "maximum port weight" β used by Melen and Turner [MT89].

One way of defining a nonblocking network is given below. It is based on a definition given by Pippenger [Pip82].

Consider a game between two players called the *blocker* and *nonblocker*. The parts of the game which are fixed and cannot be changed by either player are (1) a logical graph LOG and termination capacities $\beta : V \rightarrow \mathcal{N}$; (2) a deterministic route selection algorithm A which, given the network description above, a state s, and a single connection request q, produces a set of routes R. For each $r \in R$, r must realize q and $s \cup \{r\}$ must be a compatible state for LOG. All of this is known completely by both players. The blocker will try to add and remove connection requests so that the nonblocker cannot realize the requests.

At the start of the game, both the set of connection requests REQ and the state s are empty. The blocker has the first move.

At the beginning of the blocker's move, s realizes REQ. The blocker either adds a single connection request to REQ, or it removes a single request. It is then the nonblocker's move.

At the beginning of the nonblocker's move, s does not realize REQ. If a request was removed from REQ, then the nonblocker moves by removing the corresponding route from s. If a request q was added, then the nonblocker performs algorithm A given s and q, producing a set of routes R. If R is empty, then the game is over and the blocker wins. If R is not empty, then the *blocker* selects one of these routes and adds it to s. If the game is not over, then it is the blocker's move again. It is critical that the blocker is the one to choose the route from R.

If the blocker can make a sequence of moves and choices from R such that it eventually wins, then the network LOG is *blocking for routing algorithm A*, or simply *blocking*. If the nonblocker can prevent R from ever being empty, then the network is *nonblocking (for routing algorithm A)*.

It should be clear that the only power the nonblocker has is given to it by the routing algorithm. The same network may be blocking for one routing algorithm, but nonblocking for another. Section 5 contains several examples of routing algorithms.

3. General and restricted problem descriptions

In the general design problem, we are given a physical graph, equipment descriptions, and traffic requirements. The object is to construct a feasible network which can satisfy those requirements as cheaply as possible. The set of logical graphs that can be embedded into a given physical graph is usually very large. Even given a single logical graph, determining whether it satisfies the traffic requirements may not be easy.

In order to study several variations on the problem, we vary the portion of the logical graph and embedding which is to be regarded as unchangeable or fixed. In both the PARTIAL-DESIGN and LINK-DESIGN problems described below, a physical graph and equipment descriptions are given, and the resulting logical graph and embedding must be feasible.

In the most general PARTIAL-DESIGN problem, an instance of the problem is an arbitrary logical graph, embedding, and traffic requirements. The goal is to find an addition of switches and links (but not terminals) to the logical graph and embedding such that the network satisfies the traffic requirements and has the lowest cost possible.

In the slightly more restricted LINK-DESIGN problem, an instance consists of a set of switches, their connecting links to terminals, and an embedding. The goal is to add links between switches in the logical graph and embed them in such a way that the resulting network satisfies the traffic requirements and has the lowest cost possible. Note that the links between switches and terminals are fixed, and each terminal is connected to exactly one switch. Thus we may assume that all traffic requirements are between switches.

The ANALYSIS problem is the most restricted. An instance specifies a complete logical graph and traffic requirements. An embedding and physical graph are not necessary. The goal is to answer "yes" if the network satisfies the traffic requirements, and "no" otherwise.

In the case of nonblocking requests, we also study some problems that are "between" LINK-DESIGN and ANALYSIS in the degree of freedom given to the algorithm. These problems are discussed in section 5.3.

In this report we restrict the general framework described above in several ways. First, assume that the physical graph has unlimited cross-sectional area available on link paths, and that there are no environmental constraints. With these restrictions, the physical graph specifies allowable link locations and link costs. Switches are assumed to have zero cost and unlimited capacity to terminate links. Links are available in arbitrary (integer) capacities, with a fixed cost per unit capacity and per unit length, possibly with an additional installation cost. Each terminal may only be connected to a single switch.

4. Routable Networks

4.1. Analysis with point-to-point single rate requirements is NP-complete

With the restricted framework given in the previous section, it is useful to restate the design and analysis problems in a direct form, leaving out details which no longer matter. This will be done for each problem examined.

Routable Network ANALYSIS with point-to-point single rate requirements

> INSTANCE: A logical graph $LOG = (V, E)$ with link capacities $cap : E \to \mathcal{N}$. A point-to-point single rate set of connection requests REQ.
> QUESTION: Is LOG routable for REQ? That is, find a state s compatible with LOG that realizes the requests REQ, or prove that none exists.

A restricted version of the problem above is one in which all link capacities must be 1 and all connection requests must have rate 1. This version will be called EDGE-DISJOINT CONNECTING PATHS.

EDGE-DISJOINT CONNECTING PATHS (EDCP)

INSTANCE: An undirected graph $LOG = (V, E)$. A set of vertex pairs $REQ = \{\{u_1, v_1\}, \ldots, \{u_k, v_k\}\}$.
QUESTION: Is there a collection of paths $\{p_1, \ldots, p_k\}$ such that p_i is a path from u_i to v_i for all $1 \leq i \leq k$ and no edge appears in more than one path?

Since EDCP is a restriction of the analysis problem, the analysis problem is NP-complete if EDCP is.

THEOREM 4.1. EDCP *is* NP*-complete.*

Proof: The proof is done by a polynomial transformation from the restriction of UNDIRECTED TWO-COMMODITY INTEGRAL FLOW [GJ79, problem ND39] in which all edge capacities are 1. This problem is called simple U2CIF by Even, Itai, and Shamir [EIS76], who proved it NP-complete. It is stated differently there, but when all edge capacities are 1, it can be stated more simply as given below.

Simple U2CIF

INSTANCE: An undirected graph $G = (V, E)$, source vertices s_1 and s_2, destination vertices t_1 and t_2, and requirements $R_1, R_2 \in \mathcal{N}$.
QUESTION: Are there R_1 paths from s_1 to t_1 and R_2 paths from s_2 to t_2 such that no edge appears in more than one path?

Note that the answer must be no if $R_1 + R_2 > |E|$, so the only interesting case is when $R_1 + R_2 \leq |E|$. Let an instance of simple U2CIF be given. The transformation will produce the EDCP instance $LOG = (V', E'), REQ$ where:

$$\begin{aligned} V' &= V \cup \{u_i, v_i : 1 \leq i \leq R_1\} \cup \{w_i, x_i : 1 \leq i \leq R_2\} \\ E' &= E \cup \{\{s_1, u_i\}, \{t_1, v_i\} : 1 \leq i \leq R_1\} \cup \{\{s_2, w_i\}, \{t_2, x_i\} : 1 \leq i \leq R_2\} \\ REQ &= \{\{u_i, v_i\} : 1 \leq i \leq R_1\} \cup \{\{w_i, x_i\} : 1 \leq i \leq R_2\} \end{aligned}$$

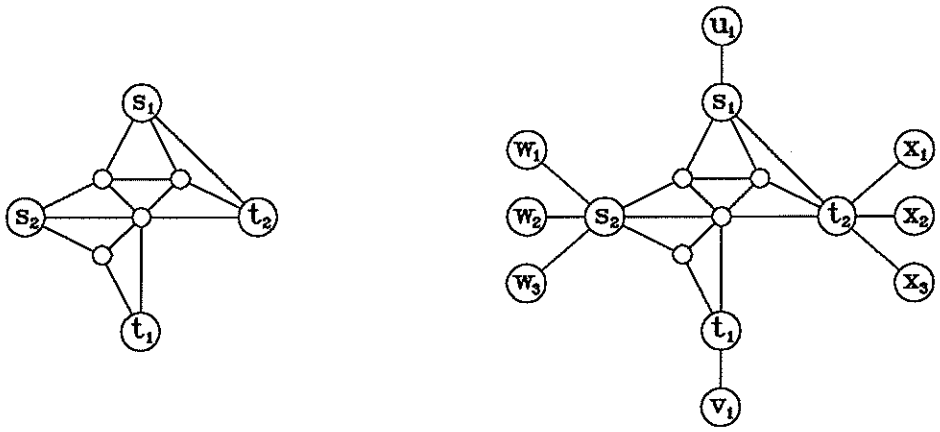
As an example of the transformation, consider the instance of U2CIF consisting of the graph in Figure 5(a) and $R_1 = 1, R_2 = 3$ (this instance has a solution). The transformation will produce the graph LOG shown in Figure 5(b) and the pairs

$$REQ = \{\{u_1, v_1\}, \{w_1, x_1\}, \{w_2, x_2\}, \{w_3, x_3\}\}$$

It is clear that the transformed instance contains a set of edge-disjoint connecting paths if and only if the original instance of U2CIF has a solution. ■

COROLLARY 4.1. *The Routable* ANALYSIS *problem with multirate traffic requirements (either point-to-point or multipoint) is* NP*-complete.*

Proof: Multirate and multipoint connection requests are generalizations of point-to-point single rate requests, so this problems are generalizations of Routable ANALYSIS with point-to-point single rate requests. Therefore it is also NP-complete. ■



(a) U2CIF graph (b) EDCP graph

Figure 5: Example of transformation from U2CIF to EDCP

4.2. Efficient Algorithm for Link-design with Link Cost Proportional to Capacity

Next we consider a problem in which we are given fixed traffic requirements, and the object is to design link capacities such that the requirements can be satisfied. Here the link costs are assumed to be proportional to the capacity assigned.

Routable Network LINK-DESIGN with link cost proportional to capacity

> INSTANCE: A logical graph $LOG = (V, E)$. For each $\{u, v\} \in E$, a cost per unit capacity $cost(u, v) \in \mathcal{N}$. The cost of building a link of capacity cap between u and v is $cost(u, v) \cdot cap$. A set of connection requests REQ.
> SOLUTION: Any function $cap : E \to \mathcal{N}$ such that if LOG has edges with those capacities, then LOG is routable for REQ.
> SOLUTION COST: $\sum_{\{u,v\} \in E} cost(u, v) \cdot cap(u, v)$
> OBJECT: Find a solution with minimum cost.

Below is a sketch of an algorithm to solve this problem efficiently.

> Create a graph $G = (V, E)$ with edge lengths given by the values $cost(u, v)$
> $cap(u, v) := 0$ for all $\{u, v\} \in E$
> for $q_i \in REQ \to$
> if $|req\text{-}terms(q_i)| = 2 \to r_i :=$ shortest path between pair of terminals
> $|\ |req\text{-}terms(q_i)| > 2 \to r_i :=$ minimum Steiner tree between set of terminals
> fi
> for $e \in r_i \to cap(e) := cap(e) + req\text{-}rate(q_i)$ rof
> rof

For the point-to-point case, only shortest paths need to be found. If all pairs of terminals have requests between them, then the algorithm above could be implemented by first finding

shortest paths between all pairs of terminals, and then processing the requests. If $n = |V|$ and $m = |E|$, then the all pairs shortest path problem can be solved in $O(n^2 \log n + nm)$ time [FT87]. This dominates the running time of the algorithm.

It is interesting to note that the point-to-point LINK-DESIGN problems are easier to solve than the corresponding ANALYSIS problems (at least for this type of link cost). Apparently the freedom to choose link capacities, as opposed to having them given, allows for a faster solution.

If instances are restricted to a single multipoint requirement, then this problem is the optimization version of the NP-complete Steiner tree problem on graphs [GJ79, problem SP12]. There are polynomial time approximation algorithms for this problem which always find a tree spanning the endpoints with cost that is no more than twice the minimum possible cost [KMB81, Wax88]. Furthermore, if there are several requirements, then such an approximation algorithm can be run for each requirement individually, and the resulting link dimensions can be summed to get an overall solution which is no worse than twice optimal.

4.3. Approximation Algorithms for Link-design with Startup Costs

We now consider the link design problem when the link costs include a startup cost. By startup costs, we mean that the cost of building a link is some given value if the link is built, and zero if it is not. The cost is independent of the link capacity. This is realistic over a large range of capacities if the labor for installation costs much more than the link itself.

Routable Network LINK-DESIGN with link cost independent of capacity

> INSTANCE: A logical graph $LOG = (V, E)$. For each $\{u, v\} \in E$, a cost to build the link $cost(u, v) \in \mathcal{N}$. A set of connection requests REQ. The rates of the requests are unimportant for this problem; each request is a set of at least two vertices.
> SOLUTION: A set of links $L \subseteq E$. For all $r \in REQ$ and all $u, v \in req\text{-}terms(r)$, there must be a path between u and v in the graph (V, L).
> SOLUTION COST: Given a set of links L, its cost is $\sum_{\{u,v\} \in L} cost(u, v)$.
> OBJECT: Find a solution with minimum cost.

Note that it is no harder to solve this problem when we allow multipoint requests, because an instance with multipoint requests can be converted into one with point-to-point requests only. If q is a multipoint request containing vertex u, then replace q with the collection of point-to-point requests $\{\{u, v\} : v \in q - \{u\}\}$. Then $L \subseteq E$ is a solution for the original instance if and only if it is a solution for the modified instance. Thus we need only consider the problem when restricted to point-to-point requests.

The decision version of this problem is NP-complete because if we restrict instances to have only one multipoint request, it is the Steiner tree problem. Agrawal, Klein, and Ravi [AKR91] have designed an approximation algorithm which runs in $O(m \log m)$ time and always finds a solution with cost at most $2(1 - 1/k)$ times optimal, where $m = |E|$ and k is the number of vertices which are contained in at least one request.

		PP/SR	PP/MR	MP/MR
ANALYSIS		NPC (Thm. 4.1)	NPC (Cor. 4.1)	
LINK-DESIGN	$c \cdot cap$	P (Sec. 4.2)		NPH2
	$s(cap > 0)$	NPH2 in general P for a restriction (Sec. 4.3)		
	$s(cap > 0) + c \cdot cap$	NPH		
PARTIAL-DESIGN		?		

Table 1: Results for routable networks

Key:
PP – point-to-point, MP – multipoint, SR – single rate, MR – multirate
P – polynomial time (efficient) algorithm known
NPC – NP-complete problem
NPH – NP-hard
NPH2 – NP-hard, but there is a known polynomial time algorithm to approximate within factor of 2 of optimal
? – currently unknown

It is useful to consider a set of point-to-point requests as defining a set of edges on the vertices V. Call the graph $RG = (V, REQ)$ the *request graph* for the instance. In general, the request graph is arbitrary. However, if we know that it consists of a single connected component and some isolated vertices, then the problem is equivalent to that of finding a minimum Steiner tree on the vertices in the connected component. There are efficient approximation algorithms for the Steiner tree problem which always find a solution with cost at most $2(1 - 1/k)$ times optimal [KMB81, Wax88]. We mention them here because they may have better performance on average than the one by Agrawal et al.

If we know that RG is connected and has no isolated vertices, then the problem is equivalent to finding a minimum spanning tree. This problem can be solved exactly in $O(m\beta(m,n))$ time, where $n = |V|$, $m = |E|$, and $\beta(m,n) \leq \log^* n$ if $m \geq n$ [FT87].

4.4. Summary of Results for Routable Networks

Table 1 presents a summary of known results. The LINK-DESIGN problem has been broken into three subcases. The notation $(cap > 0)$ is an expression which evaluates to one if the boolean expression contained in parentheses is true, and evaluates to zero if it is false. Therefore the three subcases represent, in order, (1) link cost is linear with link capacity, (2) link cost is a startup cost s if the link exists, and zero otherwise, and (3) link cost is the sum of the previous two. Note that the values of the factors c and s may vary across the links. The column headings denote restrictions on the type of fixed traffic requirements allowed. They are, from left to right, point-to-point/single rate, point-to-point/multirate, and multipoint/multirate.

Below are justifications for the table entries which were not discussed in previous sections.

LINK-DESIGN with link cost = $s(cap > 0) + c \cdot cap$

No matter what type of connection requests are given, these problems are generalizations of their counterparts with link cost = $s(cap > 0)$, and are no easier to solve. No approximation algorithms which are guaranteed to produce a solution within a constant factor of an optimum solution are known to the author. A heuristic solution based on the simplex method of linear programming is described by Gavish et al. [GTD+89].

5. Nonblocking Networks

5.1. Routing Algorithms

As mentioned in section 2.5.2, the choice of a routing algorithm can determine whether a given network is blocking or nonblocking. It can also affect the computational complexity of the analysis and design problems associated with them.

In this section we consider several examples of routing algorithms for point-to-point connection requests (some have natural extensions to multipoint requests). In each case, the algorithm knows the logical graph $LOG = (V, E)$ with link capacities cap, the current state s, and a point-to-point connection request $q = (\{u, v\}, req\text{-}rate)$. The algorithm must compute a set R of routes. Recall from the definitions that for a route r, $s \cup \{r\}$ is compatible if and only if $(\forall e \in r)\ (req\text{-}rate \leq avail(s, e))$. It is implicitly assumed that any routes not satisfying this condition are removed from the set returned.

Fixed path routing (FP)

> FP has access to a table $path$. For each pair of endpoints u, v, $path(u, v)$ is a path between them. Return $\{path(u, v)\}$. If we use this routing algorithm in an analysis or design problem, then the instance must contain the table $path$.

Fixed shortest path routing (FSP)

> This is similar to fixed path routing, except that $path(u, v)$ is restricted to be a shortest path between u and v.

Number of hops at most K (HAMK)

> HAMK returns *all* paths in LOG between u and v such that the path contains at most K links. K must be specified in the problem instance.

5.2. Properties Independent of Routing Algorithms

Before presenting complexity results, we present results which hold regardless of the routing algorithm chosen. For $X \subseteq V$, let $\overline{X}$ denote the complement of X relative to V, $V - X$. $(X, \overline{X})$ denotes a *cut*, which is the set of all edges which have one endpoint in X and the

other in $\overline{X}$. We extend the domains of *cap* and β to arbitrary sets of links and vertices by summing the individual values of each element in the set.

LEMMA 5.1. *Let A be any routing algorithm. If $LOG = (V, E)$ with link capacities cap and termination capacities β is nonblocking for A, then*

$$(\forall X \subseteq V)\,(cap(X, \overline{X}) \geq \min\{\beta(X), \beta(\overline{X})\})$$

Proof: Let A, LOG, and $X \subseteq V$ be chosen. In the game between blocker and nonblocker used to define nonblocking networks (Sec. 2.5.2), it is possible for the blocker to specify a sequence of point-to-point connection requests, each of which has one terminal in X and the other in $\overline{X}$. If each of these requests has rate one, then $\min\{\beta(X), \beta(\overline{X})\}$ such requests can be specified; no more can be added without violating the termination capacity constraints. Given such a request q, any route r realizing q must use at least one link in $(X, \overline{X})$. Therefore the total usage of links in the cut is at least $\min\{\beta(X), \beta(\overline{X})\}$. The usage can be no more than the capacity in any compatible state. ■

LEMMA 5.2. *Let $LOG = (V, E)$ be a network with link capacities cap and termination capacities β. If $(X, \overline{X})$ contains a single edge $\{u, v\}$, then for every request set REQ compatible with LOG and every state s which realizes REQ and is compatible with LOG, $usage(s, \{u, v\}) \leq \min\{\beta(X), \beta(\overline{X})\}$.*

Proof: This is easily seen by contradiction. Suppose we have a state s realizing a compatible request set REQ, and in s the usage of $\{u, v\}$ is larger than $\min\{\beta(X), \beta(\overline{X})\}$. Consider the set of routes R which use $\{u, v\}$. Each such route r has at least one endpoint in X and at least one in $\overline{X}$. Therefore it "uses up" at least $req\text{-}rate(r)$ termination capacity in X and in $\overline{X}$, and so the entire set R uses at least $usage(s, \{u, v\})$ termination capacity in X and in $\overline{X}$. But this violates the assumption that the request set REQ is compatible with LOG (by the definition of a compatible request set). ■

A consequence of this lemma is that any edge $\{u, v\}$ which is a cut $(X, \overline{X})$ by itself (commonly called a *bridge*) should have capacity exactly equal to the minimum total termination capacity on each side, $\min\{\beta(X), \beta(\overline{X})\}$. It must be at least this much for the network to be nonblocking, and no more than this is ever used regardless of whether the network is nonblocking.

Note that the proof of Lemma 5.1 only uses point-to-point single rate connection requests, thus it applies for any restriction on traffic requirements examined here. If we know that we are allowed to have multirate requests, then we can strengthen the necessary condition of Lemma 5.1. Let B be the largest rate allowed for any single connection request.

LEMMA 5.3. *Let A be any routing algorithm, and $LOG = (V, E)$ be a network with link capacities cap and termination capacities β. If multirate requests are allowed, and LOG is nonblocking for A, then*

$$(\forall X \subseteq V)\,(\max_{\{u,v\} \in (X, \overline{X})} cap(u, v) \geq \min\{\max_{u \in X} \beta(u), \max_{u \in \overline{X}} \beta(u), B\})$$

Proof: For any cut $(X, \overline{X})$, it is possible to make a request which must use at least one edge of the cut by making the request between endpoints on opposite sides of the cut. There must exist an edge in the cut which has enough capacity to carry the connection by itself. The largest possible rate for this connection can be obtained by picking the endpoints with largest termination capacity on each side of the cut. ■

5.3. Nonblocking Network Tree-design

First we consider a nonblocking network design problem in which the design algorithm must use the links given. The only freedom it has is to choose the capacities of those links. This is more restricted than the LINK-DESIGN problem.

Nonblocking network TREE-DESIGN

> INSTANCE: A logical graph $LOG = (V, E)$ which is a tree. For each $\{u, v\} \in E$, a cost per unit capacity $cost(u, v) \in \mathcal{N}$.
> SOLUTION: Any function $cap : E \rightarrow \mathcal{N}$ such that if LOG has edges with those capacities, then LOG is nonblocking for A.
> SOLUTION COST: $\sum_{\{u,v\} \in E} cost(u, v) \cdot cap(u, v)$
> OBJECT: Find a solution with minimum cost.

Recall that a tree is a connected acyclic graph, and so there is only one routing algorithm A which is reasonable to use. A returns the unique smallest subtree of LOG containing the desired endpoints.

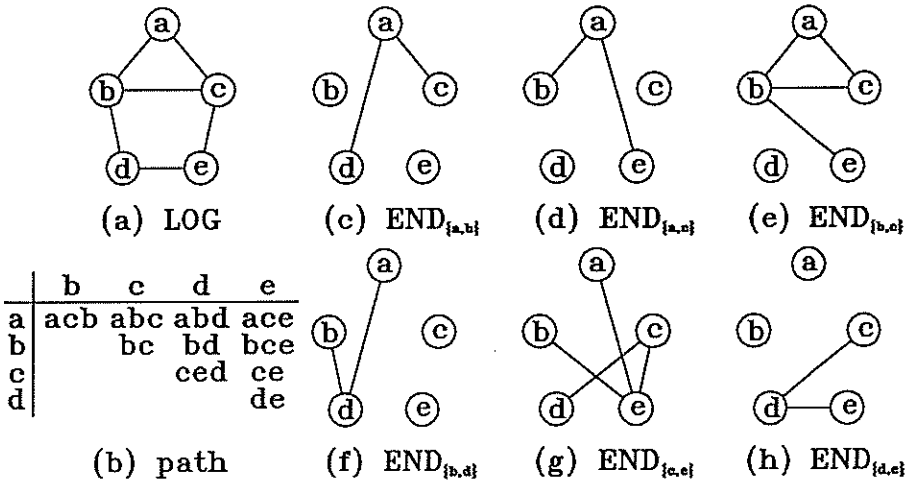
Since every edge of a tree is a cut, we can apply Lemma 5.2 to every edge. If an edge's removal leaves two connected components X and $\overline{X}$, then its capacity should be exactly $\min\{\beta(X), \beta(\overline{X})\}$. These edge capacities can be found in $O(|E|)$ time by working from the leaf vertices inward. Note that these same edge capacities will give an optimal solution even if the cost of building links is an arbitrary nondecreasing function of capacity. Also note that it does not matter whether the connection requests are single rate or multirate, point-to-point or multipoint.

5.4. Link-design with Fixed Path Routing Algorithm

For this problem, we consider only point-to-point connection requests, although it does not matter whether they are single rate or multirate.

Nonblocking network LINK-DESIGN with fixed path routing

> INSTANCE: A logical graph $LOG = (V, E)$. For each $\{u, v\} \in E$, a cost per unit capacity $cost(u, v) \in \mathcal{N}$. A fixed path table $path$ containing paths in LOG.
> SOLUTION: Any function $cap : E \rightarrow \mathcal{N}$ such that if LOG has edges with those capacities, then LOG is nonblocking for routing algorithm FP.
> SOLUTION COST: $\sum_{\{u,v\} \in E} cost(u, v) \cdot cap(u, v)$
> OBJECT: Find a solution with minimum cost.



	b	c	d	e
a	acb	abc	abd	ace
b		bc	bd	bce
c			ced	ce
d				de

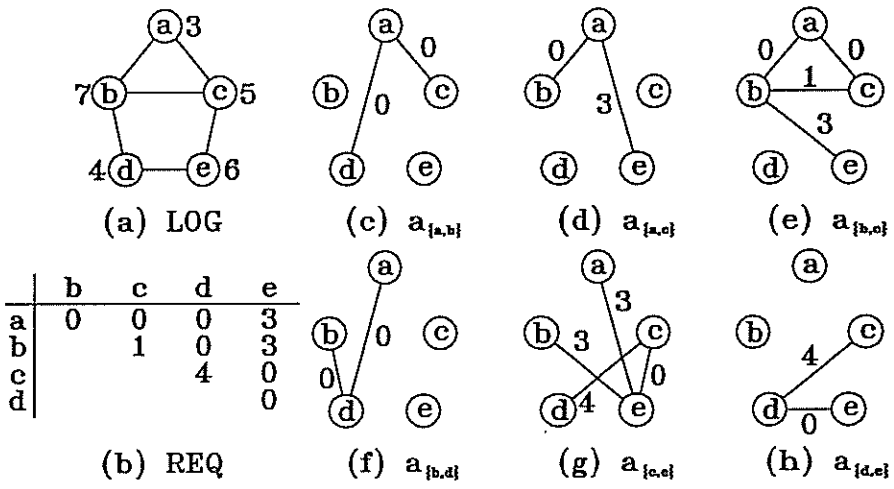
Figure 6: A logical graph and its corresponding endpoint graphs

The TREE-DESIGN problem is a special case of this one. They share an important property: given any pair of endpoints, there is a unique path which can be used to connect them. This simplifies matters by eliminating the problem of route selection. It implies that for any compatible set of requests, there is a *unique* state which realizes it. This state *does not depend* upon the link capacities, although link capacities do determine whether the state is compatible. Because of these important properties, designing the capacity of a particular link involves computing a set of requests, and hence a state, which maximizes the usage of that link. We introduce endpoint graphs to compute these requests. They allow us to transform the original design problem into $|E|$ others that have known efficient algorithms.

The *endpoint graph* of edge $\{x, y\} \in E$, denoted $END_{\{x,y\}}$, contains an edge $\{u, v\}$ if and only if $path(u, v)$ uses edge $\{x, y\}$. Figure 6 contains the endpoint graphs for the logical graph of Figure 6(a) and the fixed path routing table in 6(b).

Let REQ be a compatible set of requests and s be the unique state which results by routing all of the requests using FP. To this state, there corresponds unique *assignments* to all of the endpoint graphs, where an assignment is a mapping from the edges of an endpoint graph to $\mathcal{N}$. For edge $\{u, v\}$ of endpoint graph $END_{\{x,y\}}$, set assignment $a_{\{x,y\}}(u, v)$ to be equal to the total rate of all requests $q_i \in REQ$ which are between vertices u and v. Define the *value* of an assignment to be the sum of the individual values assigned to each edge of $END_{\{x,y\}}$. Figure 7 shows an example of this correspondence. The integer labels on vertices in 7(a) are the termination capacities. The requests REQ are represented by the table in 7(b), where each entry is the total rate of all requests between the given pair. The other parts of the figure show the corresponding assignments.

An assignment is *compatible* if for each $u \in V$, the sum of values assigned to all edges incident to u is at most $\beta(u)$. This definition is similar to the definition of compatibility for request sets, and in fact every assignment is compatible if REQ is compatible. Also, the usage of link $\{x, y\}$ in state s is exactly the value of the assignment. Formally, $usage(s, \{x, y\}) = \sum_{e \in END_{\{x,y\}}} a_{\{x,y\}}(e)$.



	b	c	d	e
a	0	0	0	3
b		1	0	3
c			4	0
d				0

Figure 7: A state and its corresponding assignments

It is also true that for each compatible assignment $a_{\{x,y\}}$ of an endpoint graph $END_{\{x,y\}}$, there is a compatible set of requests REQ which is realized by a state s such that the usage of edge $\{x,y\}$ in state s is exactly the value of the assignment. Formally, $usage(s,\{x,y\}) = \sum_{e \in END_{\{x,y\}}} a_{\{x,y\}}(e)$.

Therefore, to find the maximum usage of a link $\{x,y\}$ in any state, find a compatible assignment for $END_{\{x,y\}}$ which has a maximum value among all compatible assignments. The capacity of link $\{x,y\}$ should be the value of this assignment. Since the assignment has maximum value, the usage of the link can never be more than this. We also know that it can be this much. By repeating this process for each link, we create a capacity function cap. If every link has a capacity at least as large as specified by cap, then the network is nonblocking. If any link capacity is less than that given by cap, then the network is blocking. As long as the link costs are nondecreasing functions of capacity, a cheapest solution to the LINK-DESIGN problem is to assign exactly the values given by cap.

So now the LINK-DESIGN problem has been reduced to finding maximum value assignments for $|E|$ endpoint graphs. Formally, this problem is:

Generalized Maximum Matching (GMM)

> INSTANCE: A graph $G = (V,E)$ with termination capacities $\beta : V \to \mathcal{N}$.
> SOLUTION: An assignment $a : E \to \mathcal{N}$ which is compatible, i.e., $(\forall u \in V)$ $(\sum_{\{u,v\} \in E} a(u,v) \leq \beta(u))$.
> SOLUTION VALUE: $\sum_{\{u,v\} \in E} a(u,v)$
> OBJECT: Find an assignment with maximum value.

This problem is called generalized maximum matching because if we restrict instances to $\beta(u) = 1$ for all $u \in V$, then it is the maximum cardinality matching problem [Tar83, Chap. 9]. It is possible to reduce GMM to maximum cardinality matching, but it requires time polynomial in the β values as well as $|V|$ and $|E|$, which can be exponential in the size of the original instance. Fortunately Gabow [Gab83] has designed an algorithm to

solve the problem in $O(nm \log n)$ time, where $n = |V|$ and $m = |E|$. He calls it the upper degree-constrained subgraph problem (UDCS). The overall nonblocking network link design problem can then be solved in $O(mnU \log n)$ time, where U is the largest number of edges in any endpoint graph. $U \leq n(n-1)/2$ and it may be that large depending on the fixed routes given by *path*.

The problem which Gabow's algorithm actually solves is more general than GMM. UDCS allows one to specify an upper bound $\mu(u,v)$ on $a(u,v)$ for each edge $\{u,v\} \in E$. To solve the problem GMM above, we merely set $\mu(u,v) = \min\{\beta(u), \beta(v)\}$ for every edge.

However, since UDCS is more general, we can use this to solve a more general LINK-DESIGN problem. Suppose we have a pair of terminals u and v which have high termination capacities, say 1000 each. Then the path $path(u,v)$ must have capacity at least 1000 on every edge. If we know that many connection requests will never be made between u and v, but only 150 at most, then we can set the UDCS upper bound to $\mu(u,v) = 150$ for every endpoint graph in which edge $\{u,v\}$ appears. In general, we can specify such an upper bound for every pair of terminals. The running time of the design algorithm is not affected.

This design algorithm can also be used to design hierarchical networks. For example, suppose several subnetworks are connected together by a backbone network, and each subnetwork has a single "gateway" switch through which all connections having destinations outside the subnetwork are routed. If all terminals simultaneously requested to connect to terminals outside of their subnetwork, then this would require very large capacity links on the backbone network. Normally, most requests would be to other terminals within the subnetwork. Suppose the network designer knew that no more than a total rate U_s of connection requests would pass through the gateway from subnet s to other subnets. Then the above design algorithm could be run on subnet s by itself (without the backbone and other subnets) after adding U_s to the termination capacity of its gateway switch. The capacity of the backbone links can then be designed by running the design algorithm on the backbone network (with subnet gateway switches included) where each subnet gateway has termination capacity U_s. This procedure could be continued for any number of levels in the hierarchy.

5.5. A Special Case of Fixed Path Routing

A bipartite graph is a graph $G = (V,E)$ whose vertex set can be partitioned into two sets X and $\overline{X}$ such that every $e \in E$ has one endpoint in X and the other endpoint in $\overline{X}$. If the endpoint graph is bipartite, then the generalized maximum matching problem can be solved efficiently by transforming it to a maximum flow problem. An instance consists of a directed graph $D = (V,A)$ with distinguished source vertex s and sink vertex t. Every arc $a \in A$ has a nonnegative capacity $cap(a)$. The transformation is similar to one given by Ford and Fulkerson [FF64].

Let $G = (V,E)$, $\beta : V \to \mathcal{N}$ be an instance of GMM, where G has a bipartition given by $X, \overline{X}$. We describe how to create $D = (V', A)$ and $cap : A \to \mathcal{N}$. Let $V' = V \cup \{s,t\}$ and $A = \{(s,u) : u \in X\} \cup \{(u,v) : u \in X, v \in \overline{X}\} \cup \{(v,t) : v \in \overline{X}\}$. The capacity of any arc incident to s or t is the termination capacity of the other endpoint. The capacity of

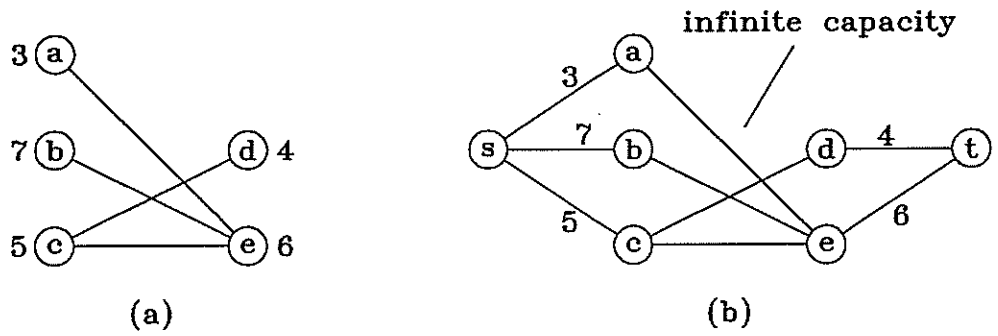


Figure 8: Transformation from bipartite GMM to maximum flow

any arc with both endpoints in V is infinity. An example of this transformation is shown in Figure 8.

This transformation will be useful for bipartite endpoint graphs, but for arbitrary fixed paths used by FP, we cannot ensure that all of the endpoint graphs will be bipartite. If we restrict all of the paths to be shortest paths, then we can show that all of the endpoint graphs are bipartite. We shall use the notation $dist_G(u, v)$ to denote the length of a shortest path between the vertices u and v in graph G with edge lengths l. The graph G intended will normally be understood from the context.

DEFINITION 1. *Let $G = (V, E)$ be an undirected graph with edge lengths $l : E \to \mathcal{N}$, and $\{x, y\} \in E$. The* shortest path endpoint graph *of edge $\{x, y\}$ in G, denoted SP-END$_{\{x,y\}}$, is defined as*

$$SP\text{-}END_{\{x,y\}} = (V, E_{\{x,y\}})$$

where

$$E_{\{x,y\}} = \{\{u, v\} : (\exists p)(p \text{ is a shortest path between } u \text{ and } v, p \text{ contains } \{x, y\})\}$$

Please note carefully the difference between shortest path endpoint graphs and endpoint graphs as given earlier. A normal endpoint graph is derived from LOG and a fixed path table, which contains a *single* path for each pair of endpoints. A shortest path endpoint graph is similar, but it includes edges for *all* shortest paths between pairs of endpoints.

LEMMA 5.4. *Let $G = (V, E)$ be an undirected graph with edge lengths $l : E \to \mathcal{N}$. For any $\{x, y\} \in E$ such that $l(x, y) > 0$, the shortest path endpoint graph SP-END$_{\{x,y\}}$ is bipartite.*

Proof: Let $u \in V$ be an arbitrarily chosen vertex. The following inequalities must hold by properties of shortest paths.

$$dist(u, y) \leq dist(u, x) + l(x, y) \quad (1)$$
$$dist(u, x) \leq dist(u, y) + l(y, x) \quad (2)$$

Since G is undirected, the edge lengths are symmetric and $l(x,y) = l(y,x)$. We shall say that an inequality is *strict* if it still holds after replacing $\leq$ with $<$, and that it is *tight* if it still holds after replacing $\leq$ with $=$. Let

$$\begin{aligned}
\mathcal{W} &= \{u \in V : \text{both (1) and (2) are tight}\} \\
\mathcal{X} &= \{u \in V : \text{(1) is tight and (2) is strict}\} \\
\mathcal{Y} &= \{u \in V : \text{(1) is strict and (2) is tight}\} \\
\mathcal{Z} &= \{u \in V : \text{both (1) and (2) are strict}\}
\end{aligned}$$

Since $l(x,y)$ is strictly positive, it cannot be that both of the inequalities are tight. Therefore the set $\mathcal{W}$ must be empty. We will demonstrate that the two sets $(\mathcal{X} \cup \mathcal{Z})$ and $\mathcal{Y}$ form a bipartition.

Next consider an arbitrary vertex $u \in \mathcal{Z}$. Let p be any shortest path from u to another vertex v. Assume that it contains the edge $\{x,y\}$. Without loss of generality, assume that x is visited before y when traversing p from u to v (the alternate case in which y is visited before x can be handled similarly). Let p_1 be the portion of p from u to x and p_2 be the portion of p from y to v. Also let q_1 be a shortest path from u to y and $q = q_1 p_2$. Then

$$\begin{aligned}
length(p) &= length(p_1) + l(x,y) + length(p_2) && \{ \text{ definition of } p \ \} \\
&= dist(u,x) + l(x,y) + length(p_2) && \{ \ p_1 \text{ is a shortest path } \} \\
&> dist(u,y) + length(p_2) && \{ \ (1) \text{ is strict } \} \\
&= length(q_1) + length(p_2) && \{ \text{ definition of } q_1 \ \} \\
&= length(q) && \{ \text{ definition of } q \ \}
\end{aligned}$$

Therefore, q is a path from u to v which is shorter than p. But p is a shortest path from u to v, so we have a contradiction, and our assumption that p contained $\{x,y\}$ must be false.

The conclusion is that there can be no shortest path from $u \in \mathcal{Z}$ to any other vertex such that the path contains the edge $\{x,y\}$. Therefore, u must be isolated in $SP\text{-}END_{\{x,y\}}$.

By a similar "short-cutting" method as used above, it can be shown that no two vertices $u, v \in \mathcal{X}$ can have a shortest path between them which contains $\{x,y\}$. Therefore u and v will not have an edge between them in $SP\text{-}END_{\{x,y\}}$. Also, any pair of vertices in $\mathcal{Y}$ may not be adjacent in $SP\text{-}END_{\{x,y\}}$.

So the two sets $(\mathcal{X} \cup \mathcal{Z})$ and $\mathcal{Y}$ form a bipartition for the graph $SP\text{-}END_{\{x,y\}}$. ■

The fastest solution to the maximum flow problem known to the author was given by Goldberg and Tarjan [GT88]. It runs in $O(nm\log(n^2/m))$ time, which is asymptotically better than Gabow's running time for GMM. The overall running time would be $O(mnU\log n)$ as before, although here $U \leq n^2/4$. It may be possible to use the results of Gusfield et al. [GMFB87] to improve this bound. The author believes that shortest path routing would arise often enough for this bipartite special case to be interesting, and someone wishing to implement this algorithm may have access to an implementation of a maximum flow algorithm, whereas it is doubtful they would already have an implementation of an algorithm for GMM.

At the end of the previous section, we described a generalization of the design problem in which we can specify an upper bound $\mu(u,v)$ on the total rate of connection requests

	FSP	FP	HAMK
ANALYSIS	P	P	co-NP
TREE-DESIGN	P		
LINK-DESIGN	P	P	?

Table 2: Results for nonblocking networks

between particular pairs of terminals. The same restriction can be implemented in the reduction to maximum flow instances given above. If arc (u, v) or (v, u) appears in a maximum flow instance, then set its capacity to $\mu(u, v)$ instead of infinity.

5.6. Summary of Results for Nonblocking Networks

A summary of complexity results is given in Table 2. On the left are the problems examined in the context of nonblocking requirements. On the top are the routing algorithms for which we have found complexity results: fixed shortest path, fixed path, and number of hops at most K. The table entries contain the complexity of the problem. See the key of Table 1 for their meanings. The ANALYSIS results for routing algorithms FP and FSP are explained by noting that the analysis problem can be solved by first solving the LINK-DESIGN problem, and then simply checking whether all of the given link capacities are at least as large as the optimum capacities.

For TREE-DESIGN, there is really only one routing algorithm, which could be thought of as FSP. In fact, it could be solved by the algorithm given in section 5.5, but that algorithm is slower than the one in section 5.3.

ANALYSIS with routing algorithm HAMK is in co-NP means that given an instance of the problem, asking the opposite question "Is LOG blocking for algorithm HAMK?" results in a problem that is in NP. To see that the negated problem is in NP, all that is necessary is to nondeterministically guess a set of requests REQ and a state s which realizes it. The deterministic part of the algorithm then verifies that there is some pair of vertices u and v which both have available termination capacity, but for which there is no path of length at most K with available capacity between them.

6. Conclusion

The author plans to continue exploring computational complexity questions and searching for efficient algorithms or approximation algorithms. One concern is that fixed path routing requires high link capacities. To that end, the author hopes to reduce these requirements by studying more flexible routing algorithms (but not as flexible as HAMK). The following are two ideas for further investigation.

Deflection routing (DEF)

DEF looks up u and v in a table as FP does, but in this case the result is two paths p_1 and p_2, called the primary path and secondary path, respectively. DEF checks the current state s to see if p_1 has enough capacity available. If so, $\{p_1\}$ is returned, otherwise $\{p_2\}$ is returned.

Shortest available path routing (SAP)

Consider the set of links $E' = \{e : e \in E \wedge \textit{req-rate} \leq \textit{avail}(s, e)\}$, i.e., E' contains exactly those links which have enough available capacity to be usable in a route. SAP returns all paths which have shortest length in the graph (V, E').

The author would like to thank Rex Hill, Buddy Waxman, and Ellen Witte for their comments on previous versions of this paper. He is also grateful to Jon Turner for his guidance and discussions involved in this work.

References

[AKR91] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. In *23rd Symp. Theory of Computing*, pages 134–144, 1991.

[dP91] M. de Prycker. *Asynchronous transfer mode: Solution for broadband ISDN.* Ellis Horwood, 1991.

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, December 1976.

[FF64] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ, 1964.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[Gab83] Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proc. 15th Annual ACM Symp. on Theory of Computing*, pages 448–456, 1983.

[GJ79] Michael R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to NP-Completeness.* Freeman, 1979.

[GK90] R. J. Gibbens and F. P. Kelly. Dynamic routing in fully connected networks. *IMA J. Mathematical Control and Information*, 7:77–111, 1990.

[GMFB87] D. Gusfield, C. Martel, and D. Fernandez-Baca. Fast algorithms for bipartite network flow. *SIAM J. Comput.*, 16:237–251, 1987.

[GN89] Bezalel Gavish and Irina Neuman. A system for routing and capacity assignment in computer communication networks. *IEEE Trans. Comm.*, 37(4):360–366, April 1989.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, October 1988.

[GTD+89] B. Gavish, P. Trudeau, M. Dror, M. Gendreau, and L. Mason. Fiberoptic circuit network design under reliability constraints. *IEEE J. Sel. Areas Commun.*, 7(8):1181–1187, October 1989.

[GW90] A. Gersht and R. Weihmayer. Joint optimization of data network design and facility selection. *IEEE J. Sel. Areas Commun.*, 8(9):1667–1681, December 1990.

[Hui90] Joseph Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks.* Kluwer, 1990.

[KKG91] A. Kershenbaum, P. Kermani, and G. Grover. MENTOR: An algorithm for mesh network topological optimization and routing. *IEEE Trans. on Commun.*, 39(4):503–513, July 1991.

[KMB81] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[MT89] Riccardo Melen and Jonathan S. Turner. Nonblocking multirate networks. *SIAM J. Comput.*, 18(2):301–313, April 1989.

[Pip82] Nicholas Pippenger. Telephone switching networks. In Stefan A. Burr, editor, *The Mathematics of Networks (Proc. of symposia in applied mathematics - v. 26)*, pages 101–133. American Mathematical Society, 1982.

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms.* SIAM, 1983.

[Wax88] Bernard M. Waxman. Routing of multipoint connections. *IEEE J. Sel. Areas Comm.*, 6(9):1617–1622, December 1988.