ORNL/TM-2004/3

Computing Path Tables for Quickest Multipaths In Computer Networks

January 2004

William C. Grimmell and Nageswara S. V. Rao
Computer Science and Mathematics Division
Oak Ridge National Laboratory

ORNL/TM-2004/3

COMPUTING PATH TABLES FOR QUICKEST MULTIPATHS IN COMPUTER NETWORKS

William C. Grimmell and Nageswara S.V. Rao

Date Published: January 2004

Prepared by
OAK RIDGE NATIONAL LABORATORY
P.O. Box 2008
Oak Ridge, Tennessee 37831-6285
managed by
UT-Battelle, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

We should like to thank Professor Guoliang Xue of Arizona State University for his insights that we were fortunate to receive as we considered various aspects of the work described in this report.

The support of this work by Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, the Engineering Science Program and High-Performance Networks Program of the Office of Science, Department of Energy, the Advanced Networking Infrastructure Program of National Science Foundation, and Network Modeling and Simulation Program of Defense Advanced Research Projects Agency is acknowledged by the authors.

ABSTRACT

We consider the transmission of a message from a source node to a terminal node in a network with n nodes and m links where the message is divided into parts and each part is transmitted over a different path in a set of paths from the source node to the terminal node. Here each link is characterized by a bandwidth and delay. The set of paths together with their transmission rates used for the message is referred to as a multipath. We present two algorithms that produce a minimum-end-to-end message delay multipath path table that, for every message length, specifies a multipath that will achieve the minimum end-to-end delay. The algorithms also generate a function that maps the minimum end-to-end message delay to the message length. The time complexities of the algorithms are $O(n^2((n^2/\log n) + m)\min(D_{max}, C_{max}))$ and $O(nm(C_{max} + n\min(D_{max}, C_{max}))$ when the link delays and bandwidths are non-negative integers. Here D_{max} and C_{max} are respectively the maximum link delay and maximum link bandwidth and C_{max} and D_{max} are greater than zero.

Keywords and Phrases: Route computation algorithms, quality of service, maximum flow methods, multipaths.

1. INTRODUCTION

This report presents pseudopolynomial time algorithms that generate a "path table" of "quickest multipaths" for a network. The path table consists of multipaths and their associated message length ranges. Each multipath is a minimum end-to-end delay multipath for any message length in its associated range, and the ranges are intervals that intersect only at their end points and cover $(0, \infty)$. As part of the process of generating the path table, the algorithms also generate the inverse of the function that maps message length to minimum end-to-end delay.

Determining the maximum length message that can be transmitted within a given time is equivalent to solving a problem analyzed by Ford and Fulkerson (Ford and Fulkerson 1962) and referred to therein as a general minimal cost flow problem. The Ford-Fulkerson approach can be readily extended to find a solution to the well-known minimum cost maximum flow problem. Edmunds and Karp (Edmunds and Karp 1972) provided two approaches to this problem. The first approach closely follows Ford and Fulkerson and is applicable to aspects of quickest multipath path table generation[1]. Xue et. al. (Xue et. al. 1998) noted that the Ford and Fulkerson general minimal cost flow approach can be adapted to the problem of finding a quickest multipath for a message length. However they didn't provide an algorithm that generally determines the function that maps message length to minimum end-to-end delay and produces minimum end-to-end delay multipaths[2].

A minimum time multipath problem, **MTMPP**, is a problem of choosing a set of "simple paths" in a network between a transmitting node and a receiving node, transmission rates for each path in the set and an assignment of parts of a message to each path in the set such that the message is completely received in the minimum possible time, i.e., has minimum end-to-end multipath delay. A network, $N = (G(V, E), C, D)$ is specified by a directed graph $G(V,E)$ with nodes $v \in V$ and links $(u, v) \in E$ (where $u, v \in V$ with $u \neq v$), a capacity (bandwidth) function $C:E \rightarrow (0,\infty)$ and a delay function $D: E \rightarrow [0, \infty)$, where $v \in V \Rightarrow \exists\ u \in V$ such that (u, v) and/or $(v, u) \in E$[3]. No more than $C(u, v)$ message units per unit time can be transmitted along (i.e., flow along) the

[1] Ford and Fulkerson illustrated how their general minimal cost flow problem can be formulated as a linear program problem and then used quantities comparable to linear programming "dual variables" in their approach, which includes a series of network flow maximizations (see Appendix A, Subsection A.4). Edmunds and Karp's approach directly solves a series of minimum delay path problems on the way to a single minimum cost maximum network flow (see Appendix A, Subsection A.5). However, in the derivation of one bound on the number of minimum delay paths they require, they utilize the concept of a series of flow maximizations. Our method utilizes elements of both approaches.

[2] A polynomial time algorithm, whose development began from results of Ford and Fulkerson (Ford and Fulkerson 1962), was presented by Xue (Xue et. al. 1998) and claimed to achieve the results of the algorithm of this report. That algorithm however is not universal as shown by a counter example in this report. The Xue et. al. algorithm was also considered by Rao and Batsell (Rao and Batsell 1998) without benefit of the Ford Fulkerson results. However its lack of universality was pointed out to them in a correspondence from Grimmell. Latter in a correspondence with Xue it was found that he had independently recognized the "suboptimality" of the algorithm. Subsequently Xue (Xue 2003) created a polynomial time algorithm that finds a multipath for a given message length such that the multipath's end-to-end delay has a least integer upper bound for the message length (see Appendix C, Subsection C.2).

[3] A network might be more broadly defined than is done here, e.g., in algorithms presented in this report we shall deal with networks which have one or more zero capacity links, one or more infinite capacity links, one or more links with infinite delay, and one or more isolated nodes (i.e., $v \in V$ such that $\forall\ u \in V$, $(u, v) \notin E$ and $(v, u) \notin E$). In Section 3, a "residual network" containing links in which flow can be in either direction is defined. However we shall always assume that we start with a network in which all link capacities are > 0 and $< \infty$, all the link delays are ≥ 0 and $< \infty$, and no isolated nodes exist. We shall also assume that our starting network will have no more than one link per ordered node pair that can carry flow from the first to the second node. However, a situation in which there are a greater number of links from the first to the second node of an ordered node pair can be transformed into the one link model by using dummy nodes, a.k.a. fictitious nodes, to distinguish the multiple links.

link *(u,v)* from *u* to *v* and any part of the message (e.g., its leading edge) will arrive at *v, D*(*u,v*) time units after it is transmitted from *u*.

A path $P = [v_1, v_2, ..., v_p] = (v_1, v_2), (v_2, v_3), ..., (v_{p-1}, v_p)$ in *N* consists of a sequence of nodes $v_1, v_2, ..., v_p$ and links between the nodes where $(v_i, v_{i+1}) \in E \ \forall \ i : i = 1, 2, ..., p-1$. A path is a simple path if $v_i \neq v_j \ \forall \ i,j \ \ i \neq j$. A generalized path $GP = [[u_1, u_2, ..., u_q], [d_2,d_3, ..., d_q]] = (u_1, u_2, d_2), (u_2, u_3, d_3), ..., (u_{q-1}, u_q, d_q)$ in *N* consists of a sequence of nodes $u_1,u_2, ..., u_q$ and links between the nodes where the links are designated by a set of directions $[d_2,d_3, ..., d_q]$ and each d_i is either 1 or -1. If $d_i = 1$ then $(u_{i-1}, u_i) \in E$ is in the generalized path and if $d_i = -1$ then $(u_i, u_{i-1}) \in E$ is in the generalized path. We shall refer to a link of a generalized path with $d_i = 1$ as a forward flow link, a *ffl*, and a link with $d_i = -1$ as a counter flow link, a *cfl*. Note that every path can be considered a generalized path with all positive link directions, but any generalized path with a negative link direction is not equivalent to a path[4]. A generalized path is a simple generalized path if $u_i \neq u_j \ \forall \ i,j \ \ i \neq j$.

A path's delay is defined as the sum of the path's link delays, i.e.,

$$D([v_1, v_2, ..., v_p]) = \sum_{i=1}^{p-1} D(v_i, v_{i+1}) \tag{1}$$

The *shortest path* from node *u* to node *v* in a network is a path that has the minimum path delay of all paths from *u* to *v* in the network. A path capacity (bandwidth) is defined as the minimum of the path's link capacities, i.e.,

$$C([v_1, v_2, .. v_p]) = \min_{i=1,...,p-1} C(v_i, v_{i+1}) \tag{2}$$

It is assumed that a message sent over a path $[v_1, v_2, ..., v_p]$ would be transmitted at a rate $R \leq C([v_1, v_2, ..., v_p])$ message units per unit time along each link (v_i , v_{i+1}) (leading to no buffering of parts of the message at any node). If the leading edge was transmitted from v_1 at time $T = 0$ along the path $[v_1, v_2, ..., v_p]$, then the trailing edge would arrive at v_p at time:

$$T = D([v_1, v_2, .. v_p]) + \frac{\sigma}{R} \tag{3}$$

where σ is the message length. The maximum length message that can be sent over a path as a function of time *T* is then:

$$\sigma_{\max} = C([v_1, v_2, .. v_p])(\mathrm{T} - D([v_1, v_2, ..., v_p])) \tag{4}$$

The *end-to-end delay* for a message of length σ over a path $[v_1, v_2, ..., v_p]$ is:

$$T = D([v_1, v_2, .. v_p]) + \frac{\sigma}{C([v_1, v_2, ..., v_p])} \tag{5}$$

i.e., the time between transmission of the leading edge of the message from node v_1 and the reception of its trailing edge at node v_p if the transmission rate is the path capacity (and the message is transmitted continuously). We will refer to a *quickest path* from a node v_1 to node v_p in a network for a message of length σ as a path from v_1 to v_p in the network that gives the minimum end-to-end delay for the message for all network paths from v_1 to v_p.

[4] The term "chain" is used by Ford and Fulkerson (Ford and Fulkerson 1962) to designate what we call a "path" and the term "path" to designate what we call a "generalized path". We have chosen our terminology because our use of "path" is in keeping with certain current networking vernacular. Terminology differences apparently are what led to the claim of greater generality than warranted for Xue et.al.'s result (Xue et. al. 1998).

A MTMPP is formally defined as follows: Given a message of length $\sigma > 0$, to be transmitted in a network $N=(G(V,E), C,D)$ from s to t where $s,t \in V$, choose a set of simple paths $P = (P_1, P_2, ..., P_p)$ from s to t, an assignment of message units $L(P_i)$ to each path P_i, and an assignment of path transmission rates $R(P_i)$ to each path P_i where:

$$\sum_{P_i \in P} L(P_i) = \sigma \tag{6}$$

$$\sum_{P_i:(u,v)\in P_i} R(P_i) \le C(u,v) \qquad \forall (u,v) \in E \tag{7}$$

such that starting to simultaneously transmit $L(P_i)$ message units at a rate $R(P_i)$ from s to t over each path P_i leads to the full message being received at t in the minimum time from the initiation of the transmission.

We define a multipath as a set of simple paths with common initial and final nodes and an assignment of rates to the paths per inequality (7) above. Note that for any multipath $MP = (P, R) = ((P_1, P_2, ..., P_p), (R(P_1), R(P_2), ..., R(P_p)))$ and any message of length σ such that $\sigma \ge \sum_{i=1}^{p} R(P_i)((\max_{j=1,2,...,p} D(P_j)) - D(P_i))$, the message can be transmitted over the multipath in the minimum time of $T_{MP,\sigma}$ where:

$$T_{\mathrm{MP},\sigma} = \frac{\sigma + \sum_{P_i \in P} R(P_i)D(P_i)}{\sum_{P_i \in P} R(P_i)} \tag{8}$$

if the message segments assigned to each path in the multipath are[5]:

$$L(P_i) = R(P_i)\left(\frac{\sigma + \sum_{P_j \in P} R(P_j)D(P_j)}{\sum_{P_j \in P} R(P_j)} - D(P_i) \right) \tag{9}$$

We shall call $T_{MP,\sigma}$ the end-to-end multipath delay for a message of length σ sent from s to t over multipath MP. An MTMPP with these definitions can be stated as a problem in which a network, a starting node, a terminal node and a message length are given, and a network multipath from the starting to terminal node with a minimum end-to-end delay, i.e., a *quickest multipath*, for the message is to be found.

We shall define the capacity, $C(P,R)$, of a multipath (P,R) to be:

$$C(P,R) = \sum_{P_i \in P} P(P_i) \tag{10}$$

and define (somewhat arbitrarily) the delay, $D(P,R)$, of the multipath (P,R) to be:

$$D(P,R) = \frac{\sum_{P_i \in P} R(P_i)D(P_i)}{\sum_{P_i \in P} R(P_i)} \tag{11}$$

This allows the length of a maximum length message that can be sent over the multipath in a time T to be expressed as $\sigma_{max} = C(P,R)(T-D(P,R))$ which is analogous to the expression for the maximum length message

[5] Here we assume the message can be divided into segments of arbitrary positive length neglecting that there may be a minimum length message unit. The effects of having to make the $L(P_i)$ integer are considered in Appendix C. One effect, as shown by Kargaris et. al. (Kargaris et. al. 1999), is the MTMPP becomes NP-complete.

that can be transmitted over a path in time T. The end-to-end delay time for a message of length σ over a multipath (P,R) can then be expressed as:

$$T = \frac{\sigma}{C(P,R)} + D(P,R) \tag{12}$$

Finding a shortest path in a network can be accomplished by a number of well-known algorithms. Dijkstra's algorithm, see e.g., Cormen et. al. (Cormen et. al. 1990), implemented with Fibonnaci heaps accomplishes this with complexity[6] $O(m + n\log n)$, where here n is the number of elements in V, the set of network nodes, and m the number of elements in E the set of network links. Dijkstra's algorithm does not however operate in a more general situation in which negative link delays are allowed in a network (a situation potentially applicable to the algorithms that we shall discuss in Sections 4 and 5). However the Bellman-Ford algorithm (see e.g., the Cormen et. al. reference) can, in the absence of "negative delay loops", handle this more generalized situation and has complexity $O(nm)$.

Rao and Batsell (Rao and Batsell 1997) among others have shown that the quickest path delay for a message of length σ from a node s to a node t in a network is a piecewise linear function of σ where each linear piece represents a particular path (or any one of a set of paths) in the network from s to t and where these paths have greater delay and increased capacity as σ increases. Similarly, as has been noted by Xue et. al. (Xue et. al. 1998), the Ford Fulkerson presentation (Ford and Fulkerson 1962) can be used to show that the quickest multipath delay from s to t in a network with integral capacities and delays is a piecewise linear function of the message length σ with monotonically decreasing slope. Each linear portion can be associated with a multipath (or any one of a set of multipaths), where these multipaths have greater delay and increased capacity as σ increases. This function can be found by finding its inverse[7] and, given aspects of such an inverse, major attention will be paid to it in the sections below.

---

[6] When we use the term complexity in this report we are referring exclusively to an execution time related bound expressed in terms of network parameters, i.e., to "time complexity".

[7] The function is 1 to 1 from $[0, \infty)$ onto $[D_{min}, \infty)$, where D_{min} is the delay of a shortest path, consequently it has an inverse with domain $[D_{min}, \infty)$.

2. CONJECTURE AND COUNTER EXAMPLE

The following, which we shall refer to as the Conjecture, is stated as a theorem in Xue et. al. (Xue et. al. 1998), where it is assumed that nodes $s, t \in V$ of a network $N = (G(V, E), C, D)$ where C and D's ranges are the non-negative integers[8]:

Conjecture: Let $F(T)$ be the maximum amount of data that can be transmitted from node s to t in time T, T=0,1,2,… . Then $F(T)$ is a monotonically increasing piecewise-linear function whose number of pieces is no more than m + 1 and the function $F(T)$ can be computed by repeatedly computing a shortest path, P_i, (according to delay) with positive capacity, adding the selected path to the flow[9] and taking its capacity away from the links in the original network that are in this path until there is no path from s to t with a positive capacity. For any non-negative integer T, the maximum amount of data that can be transmitted from node s to node t in T time units can be achieved by transmitting $c(P_i)$ units of data at time unit j for j=0, 1, 2, T-$D(P_i)$ -1 along path P_i, where $c(P_i)$ and $D(P_i)$ represent the capacity and delay of path P_i respectively.

We note that $c(P_i)$ refers to the capacity of the network's shortest non-zero capacity path after the reduction of each of path P_j's link capacities by $c(P_j)$ for j = 1, 2, …, i -1, which implies $c(P_i) \leq C(P_i)$ for all P_i, where $C(P_i)$ is path P_i 's capacity in the initial network.

Based on the supposed general correctness of the Conjecture, it is claimed in the reference that $F(T)$ can be computed in time $O(m^2 + mn\log n)$, since a shortest path in a network of positive delays can be computed in time $O(m + n\log n)$ and each computation of a shortest path also effectively removes one link from the network (sets its capacity to zero from which it will not be changed for the rest of the procedure).

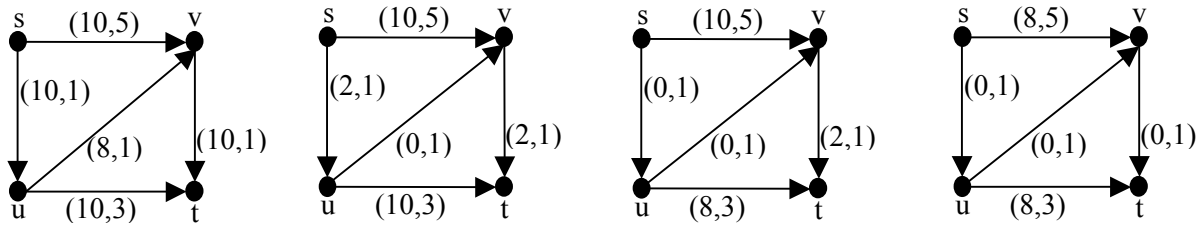

Fig. 1a Fig. 1b Fig. 1c Fig. 1d

Fig. 1. Network in which procedure of the conjecture leads to incorrect *F(T)*.

Consider now the network of Figure 1a above that has the same topology as a network used in an example in the reference but has different link capacities and delays. Here the first number in the pair by each link is the link capacity (or reduced capacities for Figures 1b to 1d) and the second is the link delay. The arrows show the forward flow direction of the links. Figures 1b through 1d show the results of applying the procedure of the Conjecture to this network. First path [*s, u, v, t*] is chosen leading to Figure 1b, then path [*s, u, t*] is chosen leading to Figure 1c and finally path [*s, v, t*] is chosen leading to Figure 1d that has no remaining non-zero capacity paths from *s* to *t*. The resulting $F(T)$ is:

[8] A zero capacity link can be considered to be non-existent so restricting C's range to the positive integers by dropping all links with zero capacity would result in an equivalent network.

[9] The phrase "adding the selected path to the flow" is not essential to the conjecture. The term flow as used in the phrase is defined below in Section 3.

$$F_1(T) = \begin{cases} 0 & T \le 3 \\ 8T - 24 & 3 \le T \le 4 \\ 10T - 32 & 4 \le T \le 6 \\ 12T - 44 & 6 \le T \end{cases} \tag{13}$$

Inspection of Figure 1d leads to the suspicion that $F_1(T)$ is not the maximum message length versus multipath end-to-end delay function since the maximum flow (i.e., the maximum sum of transmission rates along paths from *s* to *t*) for the network is 20 not 12, where the maximum flow can be achieved by using the paths (*s, v, t*) and (*s, u, t*). We confirm this suspicion by considering a message of length 160. By $F_1(T)$ a message of this length can be received in no less time than $T = 17$. However if we send 70 units via path (*s, v, t*) and 90 units via path (*s, u, t*), each at a rate of 10 message units per unit time, the message can be received in time $T = 13$. The correct value for $F(T)$ for the Figure 1 network is:

$$F_2(T) = \begin{cases} 0 & T \le 3 \\ 8T - 24 & 3 \le T \le 4 \\ 10T - 32 & 4 \le T \le 6 \\ 12T - 44 & 6 \le T \le 7 \\ 20T - 100 & 7 \le T \end{cases} \tag{14}$$

Here the linear segments of $F_2(T)$, after the $F_2(T) = 0$ segment, result successively from the following multipaths:

$8T - 24 \quad 3 \le T \le 4 \quad (([s, u, v, t]), (8))$
$10T - 32 \quad 4 \le T \le 6 \quad (([s, u, v, t], [s, u, t]), (8, 2))$
$12T - 44 \quad 6 \le T \le 7 \quad (([s, u, v, t], [s, u, t], [s, v, t]), (8, 2, 2))$
$20T - 100 \quad 7 \le T \quad (([s, u, t], [s, v, t]), (10, 10))$

The Conjecture has thus been shown to be false by the counter example of Figure 1. Interestingly the shortest path of the Figure 1a network is not a path in the quickest multipath for messages greater than 40 units, whereas if the Conjecture were correct it would remain present in all quickest delay multipaths of the network[10].

[10] The final linear segment of $F_2(T)$ might be considered to result from path (*s, u, v, t*) with rate 8, path (*s, u, t*) with rate 2, path (*s, v, t*) with rate 2 and generalized path (*s, v, u, t*) with rate 8. However since data cannot flow from *v* to *u*, (*s, v, u, t*) cannot be used as part of a data transmitting "multipath".

3. RELATIONSHIPS BETWEEN MULTIPATHS AND NETWORK FLOWS

A "flow" along a set of simple paths $P = (P_1, P_2, \ldots, P_p)$ in a network from node s to node t, where each "path flow" is at a rate $R(P_i)$ for $i = 1$ to p, can be *composed* into a "network flow" $f: E \rightarrow$ non-negative reals, where for $(u, v) \in E$:

$$f(u,v) = \sum_{P_i:(u,v)\in P_i} R(P_i) \tag{15}$$

The *magnitude* of a flow f from s to t, as represented in (15) can be defined as:

$$|f| = \sum_{u\in V:(u,t)\in E} f(u,t) = \sum_{u\in V:(s,u)\in E} f(s,u) \tag{16}$$

It follows from (15) and (16) that:

$$|f| = \sum_{P_i\in P} R(P_i) \tag{17}$$

$$\sum_{v\in V:(u,v)\in E} f(u,v) - \sum_{v\in V:(v,u)\in E} f(v,u) = 0 \qquad \forall u \in V : u \neq s,t \tag{18}$$

The sum of a multipath's path flow rates, given relation (17) is the magnitude of the corresponding network flow. The *data flow* of multipath (P, R) from s to t in time T for $T \geq \max_{P_i\in P} D(P_i)$, i.e., the length of the message that will be transmitted from s to t over (P, R) in time T can, when $f(u,v)$ is defined by (15) be expressed in terms of f as follows:

$$DF(f,T) = |f|T - \sum_{(u,v)\in E} f(u,v)D(u,v) \tag{19}$$

More generally a *network flow* from s to t satisfies (18), and

$$0 \leq f(u,v) \leq C(u,v) \qquad \forall (u,v) \in E \tag{20}$$

The magnitude of a network flow f from s to t in this more general case can be defined as

$$|f| = \sum_{u\in V:(u,t)\in E} f(u,t) - \sum_{u\in V:(t,u)\in E} f(t,u) = \sum_{u\in V:(s,u)\in E} f(s,u) - \sum_{u\in V:(u,s)\in E} f(u,s) \tag{21}$$

(where when f results from a composition of path flows per (15) the terms of the second summation in the two expressions for $|f|$ in (21) are zero[11]). A *maximum flow* from a node to another node in a network is a network flow which has the maximum possible magnitude for network flows from the first to the second node, i.e., a flow f from the first to the second node such that $|f|$ has the maximum possible value (given the network's link capacities).

We shall now state as lemmas with proofs two facts about general network flow decompositions that are relevant to our algorithms.

Lemma 1: A general network flow from s to t can be *decomposed* into a (not necessarily unique) set of three flow components (any of which can be zero) where the three components are:

1. a flow along a multipath from s to t (a forward multipath)

[11] No simple path from *s to t* has links of the form *(t, u)*. However in a more general case for a flow satisfying equation (18) with some non-zero terms of the form $f(t, u)$, the "magnitude" of the flow as defined in equation (17) can be negative (i.e., a flow magnitude is not a magnitude in the common usage of the term magnitude). In fact in general if two network flows f and g are added $|f + g| = |f| + |g|$ and if g is subtracted from f, $|f - g| = |f| - |g|$.

2. a flow along a multipath from t to s (a backward multipath)
3. a set of loop flows (which make no contribution to the network flow magnitude)

where the original network flow is the sum of compositions of the three decomposition components.

Proof: Given network flow f in N from s to t we can remove all links (u, v) for which $f(u, v) = 0$. Then we can successively choose a simple path in the remaining network from s to t, find the minimum of the link flows in the path and assign this minimum value to the path flow, reduce the flow in each path link by this path flow and then remove all path links for which the reduced flow is 0. (The reduced flow will continue to satisfy equation (18) since at each intermediate path node the sum of incoming and outgoing flows is reduced by the same amount and will continue to satisfy inequality (20) because the link flows are reduced but not to less than zero.) This process will terminate in m or less steps, since one or more links are removed at each step, leaving no paths from s to t. The set of chosen paths and assigned path flows form a (forward) multipath. Similarly starting with the network remaining after a link from the last path from s to t has been removed, we can create a (backward) multipath leaving a network with no paths from s to t or t to s.

After the construction of the two multipaths, the net flow into (i.e., the difference of the sum of flows into and sum of flows from) s and the net flow into t in the remaining network are both zero, since there are no paths from s to t or from t to s. Further, since the net flow into any other node in the network has not been changed by the multipath constructions, the net flow into any other node in the remaining network is also zero (i.e., equation (18) remains satisfied). Therefore any node with a link into it in the remaining network must have a link from it as well. Thus, if the network has any links, it must have loops, i.e., paths of the form $(u_0, u_1), (u_1, u_2), \ldots, (u_{p-1}, u_p), (u_p, u_0)$. Hence, we can successively choose a loop in the remaining network, find the minimum of the link flows in the loop and assign this minimum value to the loop flow, reduce the flow in each loop link by this loop flow and then remove all loop links for which the reduced flow is 0. Since a link is removed from the network for each loop, the network will be reduced to no links in $m - r$ or less steps, where r is the number of steps required to create the two multipaths.

Let f_f, f_b, and f_l be respectively the composition of the forward multipath flow, backward multipath flow and the flow of the set of loops from an above-described construction. Then, since for every link (u, v) in N, $f_f(u, v)$, $f_b(u, v)$ and $f_l(u, v)$ are respectively the link's flow reductions during the forward multipath, backward multipath and loop set constructions, and the flow in (u, v) is reduced to zero by these constructions, $f = f_f + f_b + f_l$. ■

Lemma 2: Let f be a network flow from s to t that for some T, $T > 0$, maximizes $DF(\cdot, T)$, defined in equation (19), and let f_f, f_b, and f_l be respectively the composition of the forward multipath flow, backward multipath flow and the flow of the set of loops from a decomposition of f. Then $f_b = 0$ and $DF(f_f, T) = DF(f, T)$. Further f_l will be zero on a network link with a non-zero delay and every forward multipath path will have a delay less than or equal to T.

Proof: Any non-zero flow which is a component of a decomposition of a network flow f adds to the magnitude of the second (i.e., the negative) term of $DF(f, T)$ unless it is non-zero only on links with zero delay in which case it will make no contribution to the second term. It can only add to $DF(f, T)$ if it increases the magnitude of f thereby increasing the first term of $DF(f, T)$. A flow of a backward multipath decreases and a loop flow doesn't effect the magnitude of $|f|$. Therefore, since f maximizes $DF(\cdot, T)$, $f_b = 0$ and f_l is zero on any non-zero delay link (since otherwise $DF(f_f, T) > DF(f, T)$). Hence, $DF(f_f, T) = DF(f, T)$. Further, if P is a forward multipath path whose delay $> T$, then contradictorily $DF(f_f^-, T) > DF(f_l, T)$, where f_f^- is the composition of the forward multipath excluding P. ■

A flow $R(P)$ along a generalized path P from s to t may be added to a non-maximum network flow f provided

$$R(P) \leq C(u, v) - f(u, v) \qquad \forall\ ffl\ (u, v) \in P \tag{22}$$

$$R(P) \leq f(u, v) \qquad \forall\ cfl\ (u, v) \in P \tag{23}$$

Here in adding a flow $R(P)$ along a generalized path P to a flow f, we add $R(P)$ to $f(u, v)$ for *ffl*s (u, v) in P and subtract $R(P)$ from $f(u, v)$ for *cfl*s (u, v) in P. A generalized path from s *to* t for which there is a $R(P) > 0$ meeting constraints (22) and (23) is a *flow augmenting generalized path* (when f is the network flow in N) since an increased flow along it will increase, i.e., augment, the magnitude of the network flow. It therefore follows that if N still contains a flow augmenting generalized path for s to t after a flow f is imposed upon it, then f isn't a maximum flow.

A *residual network*, N_f, where f is a network flow in N, has the same links and nodes as N but its links have forward and counter flow capacities, C_{ff} and C_{cf}. These capacities are defined as:

$$C_{ff}(u, v) = C(u, v) - f(u, v) \text{ for forward flow (from } u \text{ to } v) \text{ on link } (u, v) \tag{24}$$

$$C_{cf}(u, v) = f(u, v) \quad \text{for counter flow (from } v \text{ to } u) \text{ on link } (u, v) \tag{25}$$

A residual network may have links that can accommodate flow in both directions. Therefore a network flow g in a residual network N_f may be such that for a link (u, v) in N_f, $g(u, v) < 0$, in which case the flow of g in (u, v) is from v to u. The network flow g must however be such that $\forall$ (u, v) in N_f, $-C_{cf}(u, v) \le g(u, v) \le C_{ff}(u, v)$. Paths and generalized paths are defined in a residual network identically to how they are defined in Section 1. The capacity of a link (u, v) "in the direction of" a generalized path $P = [[u_0, u_1, \ldots, u_q, u_{q+1}][d_1, d_2, \ldots, d_{q+1}]]$, in a residual network is $C_{ff}(u, v)$ if (u, v) is a *ffl* and $C_{cf}(u, v)$ if (u, v) is a *cfl* in P. The delay, $\widetilde{D}(P)$ and capacity, $\widetilde{C}(P)$ of P, in the residual network are defined respectively as:

1. the difference between the sum of the delays of P's *ffl*s and the sum of the delays of its *cfl*s i.e.,

$$\widetilde{D}(P) = \sum_{(u,v)\in P_{ff}} D(u,v) - \sum_{(u,v)\in P_{cf}} D(u,v)$$

2. minimum of P's link capacities in the direction of P, i.e.,

$$\min\left(\min_{(u,v)\in P_{ff}} C_{ff}(u,v), \min_{(u,v)\in P_{cf}} C_{cf}(u,v) \right)$$

where $P_{ff} = \{(u, v): (u, v) \text{ is a } ffl \text{ in } P\}$ and $P_{cf} = \{(u, v): (u, v) \text{ is a } cfl \text{ in } P\}$. We define $(u_i, u_{i+1}, 1) = (u_i, u_{i+1})$ and $\widetilde{D}(u_i, u_{i+1}, 1) = D(u_i, u_{i+1})$ when $(u_i, u_{i+1}) \in E$ and $(u_i, u_{i+1}, -1) = (u_{i+1}, u_i)$ and $\widetilde{D}(P)(u_i, u_{i+1}, -1) = -D(u_{i+1}, u_i)$ when $(u_{i+1}, u_i) \in E$. Then in accordance with the definition 1 above, $\widetilde{D}(P) = \sum_{i=0}^{q} \widetilde{D}(u_i, u_{i+1}, d_{i+1})$. We note that a non-zero capacity generalized path, a *nzcgp*, from s to t in N_f is, with the definition 2 above, a flow augmenting generalized path (when f is the network flow in N). A *shortest generalized path* in a residual network N_f is a *nzcgp* with the minimum delay of all *nzcgp*s in N_f with the same initial and final nodes.

A *generalized loop* is a sequence of links and directions $(u_1, u_2, d_2), (u_2, u_3, d_3), \ldots, (u_{q-1}, u_q, d_q), (u_q, u_{q+1}, d_{q+1})$ whose initial and final nodes are the same, i.e., $u_{q+1} = u_1$. The delay and capacity of a generalized loop are defined analogously to that of a generalized path and a non-zero capacity negative delay generalized loop, i.e., a *nzcndgl*, is a non-zero capacity generalized loop whose link delays sum to a negative number. If u is a node on a *nzcndgl* in a residual network N_f then there is no shortest generalized path from s to u, since the delay of any generalized path P from s to u is greater than the delay of a generalized path consisting of P followed by the *nzcndgl*. If there is no *nzcndgl* in N_f, then any *nzcgp* Q from s to u with generalized loops in it has a delay that is not less than the delay of the simple generalized path from s to u resulting from removing the generalized loops from Q. Therefore, since there are a finite number of simple generalized paths in N_f from s to u, there is a shortest generalized path from s to u. The following lemma provides a property of N_f and shortest generalized paths that is of significance to a part of our algorithms.

Lemma 3: Let *f* be a network flow from *s* to *t* in *N* that maximizes *DF(·,T)* for some $T > 0$. Then N_f contains no non-zero capacity negative delay loops. If *P* is a shortest generalized path from *s* to *t* in N_f and *x* is a node in *P*, then $\delta(x) \le \widetilde{D}(P)$ where $\delta(x)$ is the delay of a shortest generalized path from *s* to *x* in N_f.

Proof: If *LOOP* is a *nzcndgl* in N_f then adding a flow around *LOOP* to *f* will contradictorily yield a $DF(\cdot, T) > DF(f, T)$ (see equation (19) on page 7)[12]. Hence, no *nzcndgl* is in N_f and therefore $\forall\, u$ in N_f $\exists$ a shortest generalized path from *s* to *u*. Let $P = [[x_0, x_1, \ldots, x_q, x_{q+1}], [d_1, d_2, \ldots, d_{q+1}]]$, where $x_0 = s$ and $x_{q+1} = t$, be a shortest generalized path in N_f where $\exists\ i,\ 1 \le i \le q$ such that $\delta(x_i) > \widetilde{D}(P)$ and $\forall\ j:\ i < j \le q+1$, $\delta(x_j) \le \widetilde{D}(P)$. Since *P* is a shortest generalized path, for *k* such that $1 \le k \le q+1$, $\delta(x_k) = \sum_{j=0}^{k-1} \widetilde{D}(x_j, x_{j+1}, d_{j+1})$ [13]. Further, since $\delta(x_i) > \widetilde{D}(P)$ and $\delta(x_{i+1}) \le \widetilde{D}(P)$ $\delta(x_i) > \delta(x_{i+1})$. Therefore (x_i, x_{i+1}, d_{i+1}) must have a negative delay and hence $(x_{i+1}, x_i) \in E$ and $f(x_{i+1}, x_i) > 0$. Since *f* maximizes $DF(\cdot, T)$ and $D(x_{i+1}, x_i) > 0$, by Lemma 2 on page 8, $f(x_{i+1}, x_i)$ must contribute to the forward multipath component of a decomposition of *f*. Therefore $\exists$ a path *Q* from *s* to *t* including (x_{i+1}, x_i) such that $\min_{(u,v)\in Q} f(u,v) > 0$. Let $P_{x_{i+1}}$ be the subpath of *P* from x_{i+1} to *t* and Q_{x_i} be the subpath of Q_{x_1} from x_i to *t*. Since *P* is a shortest generalized path in N_f, $\delta(x_i) - D(x_{i+1}, x_i) + \widetilde{D}(P_{x_{i+1}}) = \widetilde{D}(P)$. Hence, since $\delta(x_i) > \widetilde{D}(P)$, $D(x_{i+1}, x_i) > \widetilde{D}(P_{x_{i+1}})$. Further, since *Q* contains only forward flow links, $\widetilde{D}(Q_{x_i}) \ge 0$.

Consider the generalized loop starting at x_{i+1} traversing $P_{x_{i+1}}$ to *t* in the direction of $P_{x_{i+1}}$ then traversing Q_{x_1} to x_i counter to Q_{x_1}'s direction, then returning to x_{i+1} along (x_{i+1}, x_i) counter to that link's direction. This generalized loop has a non-zero capacity in N_f and its delay is $\widetilde{D}(P_{x_{i+1}}) - D(x_{i+1}, x_i) - \widetilde{D}(Q_{x_i})$ which is < 0 since $\widetilde{D}(Q_{x_i}) \ge 0$ and $D(x_{i+1}, x_i) > \widetilde{D}(P_{x_{i+1}})$ But this is a contradiction since N_f contains no *nzcndgl*s. Therefore $\forall\, x$ in *P*, $\delta(x) \le \widetilde{D}(P)$. ■

[12] An added flow around *LOOP* would not effect the first term of equation (19) but would decrease the second term (that is subtracted from the first).

[13] If this were not so then the path consisting of a shortest generalized path from *s* to x_k followed by the subpath of *P* from x_k to *t* would contradictorily have a smaller delay than *P*.

4. MULTIPATH ALGORITHM

The algorithms we have developed are particular instantiations of a generic algorithm that is an extension of a variant of an algorithm presented by Ford and Fulkerson (Ford and Fulkerson 1962). The extension provides the multipath path table. We shall in this section outline the general operation of the generic algorithm and prove that it produces the correct maximum length message vs. time function and provides multipaths to realize these maximum lengths. Then in the next section we shall provide details of the two algorithm instantiations, including where appropriate, why their detailed implementations are correct implementations of the generic approach, and we shall develop complexity bounds for the detailed implementations. The generic algorithm, shown below on this page, employs:

1. a shortest generalized path delay function $\delta : V \rightarrow [0, \infty]$
2. a "pruned network" $N_{f,\delta}$ that consists of:
   a. all nodes $u \in$ V such that $\delta(u) \leq \delta(t)$
   b. all links $(u, v) \in E$ such that u and v are in $N_{f,\delta}$ and $\delta(v) - \delta(u) = D(u, v)$

   and in which each link in $N_{f,\delta}$ has the forward and counter flow capacities of its corresponding link in N_f
3. an extension function *Extend* that extends a flow $\hat{g}$ in $N_{f,\delta}$ to a flow g in N_f by letting $g(u, v) = \hat{g}(u, v)$ for (u, v) in $N_{f,\delta}$ and $g(u, v) = 0$ for (u, v) not in $N_{f,\delta}$

The algorithm also uses a message length σ, a "previous value" of $\delta(t)$, $p\delta$, and three stacks *DEL*, *SIGMA*, and *MULTI*.

General Operation of Algorithm

1. $f \leftarrow 0$; $\sigma \leftarrow 0$; $p\delta \leftarrow 0$; $\delta(s) \leftarrow 0$; *DEL* ← empty stack;
   SIGMA ← empty stack; *MULTI* ← empty stack; Push *NILL* onto *MULTI*;
2. $\forall\ u \in V - \{s\}$, $\delta(u) \leftarrow$ the delay of shortest paths in N from s to u;
3. **while** $(\delta(t) < \infty)$ **do**
4. $\sigma \leftarrow \sigma + |f|(\delta(t) - p\delta)$; $p\delta \leftarrow \delta(t)$;
5. $\hat{g} \leftarrow$ a maximum flow from s to t in $N_{f,\delta}$; $g \leftarrow Extend(\hat{g})$; $f \leftarrow f + g$;
   MP ← the forward multipath component of a decomposition of f;
6. Push $\delta(t)$ onto *DEL*; Push σ onto *SIGMA*; Push *MP* onto *MULTI*;
7. $\forall u \in V - \{s\}$, $\delta(u) \leftarrow$ the delay of shortest generalized paths in N_f from s to u;
8. return *DEL*, *SIGMA*, *MULTI* and $|f|$;

Every generalized path from s to t in a $N_{f,\delta}$ network of the algorithm has a delay $\delta(t)$. Thus the pertinent part of $N_{f,\delta}$ may be thought of as the set of all non-zero capacity shortest generalized paths in N_f from s to t. Further, by Lemma 1 on page 7, any network flow f can be decomposed to produce a multipath from s to t. In that regard, if on the ith iteration of the algorithm's main loop, $\forall\ (u, v) \in E$, $g(u, v) \geq 0$ in N_f, then g is a network flow in N. Therefore it can be decomposed into a multipath MP_g. Then the multipath consisting of the paths and path flow rates of MP_g and the paths and path flow rates of the multipath generated at line 5 of the previous iteration is a forward multipath of a decomposition of the new value of f generated in this iteration.

We shall subsequently prove the following (in which stack entries are referred to by indices with the earliest stack entry having index 0):

Theorem: The above algorithm terminates and returns stacks and a value such that linear segments between the $(DEL[i], SIGMA[i])$, $0 \leq i \leq i_{max}$, and a final linear segment which starts at the last such ordered pair $(DEL[i_{max}], SIGMA[i_{max}])$ and continues to ∞ with a slope equal to the returned value of $|f|$ form the function $F(T)$, providing the maximum message length as a function of time. Multipath $MULTI[i]$, $0 < i \leq i_{max}$, is a

quickest multipath for message lengths between and including $SIGMA[i\text{-}1]$ and $SIGMA[i]$, and multipath $MULTI[i_{max} + 1]$ is a quickest multipath for message lengths equal to or greater than $SIGMA[i_{max}]$. The returned value $|f|$ is the maximum flow magnitude for flows from s to t in N and $DEL[i_{max}] < nD_{max}$, where $D_{max} = \max_{(u,v)\in E} D(u,v)$.

The problem of choosing a network flow f that maximizes $DF(\cdot, T)$ (of equation (19) on page 7) subject to the network constraints is a linear programming problem. Ford and Fulkerson (Ford and Fulkerson 1962) demonstrated, using linear programming duality relationships, that if a network flow f from s to t and a function π whose domain is V can be found such that:

$$\pi(s) = 0 \quad \text{and} \quad \pi(t) = T \tag{26}$$

$$\pi(v) - \pi(u) > D(u, v) \Rightarrow f(u, v) = C(u, v) \tag{27}$$

$$\pi(v) - \pi(u) < D(u, v) \Rightarrow f(u, v) = 0 \tag{28}$$

then f maximizes $DF(\cdot, T)$[14].

The following properties of shortest generalized paths are central to aspects of our proof of the theorem:

$$C(u, v) - f(u, v) > 0 \Rightarrow \delta(v) \le \delta(u) + D(u, v) \tag{29}$$

$$f(u, v) > 0 \Rightarrow \delta(u) \le \delta(v) - D(u, v) \tag{30}$$

where (as defined in Lemma 3 on page 10), $\forall x \in V$, $\delta(x)$ is the delay of the shortest generalized path from s to x[15]. If P is a shortest generalized path and $(x, y, d) \in P$ then $\delta(y) - \delta(x) = \widetilde{D}(x, y, d)$. This shortest generalized path link delay equality and the contrapositive relationships (which exist since $0 \le f(u, v) \le C(u, v)$) between conditions (27) and (29) and between conditions (28) and (30), when δ is substituted for π in equations (27) and (28), are key to the algorithm's correctness.

The algorithm generates three sequences of functions $f_0, f_1, \ldots, \hat{g}_0, \hat{g}_1, \ldots$ and $g_0, g_1, \ldots$ where $f_0 = 0$ and $f_i = f_{i-1} + g_{i-1}$ for $i > 0$, a sequence of multipaths $M_0, M_1, \ldots$ where M_0 is the null multipath, a sequence of times $\delta_0(t), \delta_1(t), \ldots$, a sequence of message lengths $\sigma_0, \sigma_1, \ldots$ where $\sigma_0 = 0$ and $\sigma_i = \sigma_{i-1} + |f_i|(\delta_i(t) - \delta_{i-1}(t))$ for $i > 0$, and two sequences of networks $N_{f_0}, N_{f_1}, \ldots$ and $N_{f_0,\delta_0}, N_{f_1\delta_1}, \ldots$, generating them in the following order: $f_i, M_i, N_{f_i}, \delta_i, \sigma_i, N_{f_i,\delta_i}, \hat{g}_i, g_i, f_{i+1}, M_{i+1}, N_{f_{i+1}}, \delta_{i+1} \ldots$. It sets $SIGMA[i]$, $DELTA[i]$ and $MULT[i]$ to σ_i, $\delta_i(t)$ and M_i respectively. The proof below of the algorithm's correctness has the following main steps:

1. A flow f_i, computed at line 5, is shown to maximize $DF(\cdot, \delta_{i-1}(t))$ and $DF(\cdot, \delta_i(t))$ over the set of possible network flows. Here $\delta_{i-1}(t)$ and $\delta_i(t)$ are the respective values of $\delta(t)$ before and after the next execution of line 7 (subsequent to the line 5 execution in which f is set to f_i). This is done by considering equations (26) - (28) with δ substituted for π and $\delta(t)$ substituted for T[16].

---

[14] As noted in the Ford and Fulkerson reference if $\gamma(u, v) = \max(0, \pi(v) - \pi(u))$, then π and γ form a solution to the dual of the linear programming problem outlined above. For a complete treatment of linear program duality as related to network general minimal cost problems see Appendix A, Subsections A.1, A.2, A.3.

[15] A shortest generalized path from s to u in N_f has non-zero capacity and has delay $\delta(u)$. Extending such a generalized path to v by appending the link (u, v), where $C(u, v) - f(u, v) > 0$, to it provides a non-zero capacity generalized path from s to v with delay $\delta(u) + D(u, v)$. Therefore since $\delta(v)$ is the delay of a shortest generalized path from s to v, $\delta(v) \le \delta(u) + D(u, v)$. Similarly extending a shortest generalized path from s to v by appending the link (u, v), where $f(u, v) > 0$, to it produces a non-zero capacity generalized path from s to u with delay $\delta(v) - D(u, v)$. Hence $\delta(u) \le \delta(v) - D(u, v)$.

[16] Since our algorithm is a variant of the algorithm provided by Ford and Fulkerson for the general minimal cost flow problem, there is a close relationship between δ and the function π used by Ford and Fulkerson. However they are not identical with δ having an obvious "physical interpretation" in the networking area that π lacks (among the differences is: the sequence of $\pi(t)$ used in the Ford and Fulkerson reference are by definition such that a successor $\pi(t)$ is greater than its predecessors while it is incumbent upon us to prove that a successor $\delta(t)$, is greater than its predecessors).

2. The sequence $\delta_0(t)$, $\delta_1(t)$, … is shown to be strictly increasing, have a finite number of elements, be non-negative, and have a final element equal to ∞.
3. $\forall\, i,\ 0 < i \le i_{max}$, f_i is shown to maximize $DF(\cdot, T)$ for T such that $\delta_{i-1}(t) \le T \le \delta_i(t)$, and $f_{i_{\max}+1}$ is shown to maximize $DF(\cdot, T)$ for T such that $\delta_{i_{\max}}(t) \le T$, where i_{max} is the index of the next to last f_i generated by the algorithm.

Proof of Theorem: Let δ be substituted for π in equations/conditions (26) – (28). Since at all times $\delta(s) = 0$, equations (26) are always satisfied for $T = \delta(t)$. Immediately after line 2 and up to the first execution of line 7, $\delta(x)$ is the delay of a shortest path for s to x in N. Hence $\forall\ (u, v) \in E$, $\delta(v) \le \delta(u) + D(u, v)$ and therefore there is no $(u, v) \in E$ such that $\delta(v) - \delta(u) > D(u, v)$. Hence condition (27) is satisfied. Since $f = 0$ from immediately after line 1 until the first execution of line 5 condition (28) is satisfied. Assume for an iteration of the algorithm loop that conditions (27) and (28) are satisfied just prior to the execution of line 5. Since all links of $N_{f,\delta}$ are such that $\delta(v) - \delta(u) = D(u, v)$, $g(u, v)$ developed at line 5 is non-zero only for (u, v) such that $\delta(v) - \delta(u) = D(u, v)$. Therefore augmenting f by g at line 5 cannot cause condition (27) or (28) to be violated. Hence the new flow, i.e. the new f, maximizes $DF(\cdot, \delta(t))$ before execution of line 7. Since the new f maximizes $DF(\cdot, \delta(t))$ no non-zero capacity negative delay loops exist in N_f (by Lemma 3 on page 10) and thus the new shortest generalized path delay function, i.e., the new δ, to be computed from N_f at line 7, exists. Therefore if $C(u, v) - f(u, v)$, the forward capacity of (u, v) in N_f, is greater than 0 (i.e., $f(u, v) \ne C(u, v)$), then for the new δ, $\delta(v) \le \delta(u) + D(u, v)$ (i.e., $\delta(v) - \delta(u) \le D(u, v)$) so condition (27) is satisfied. Also if $f(u, v)$, the counter flow capacity of (u, v) in N_f, is greater than zero (i.e., $f(u, v) \ne 0$), then $\delta(u) \le \delta(v) - D(u, v)$ (i.e., $\delta(v) - \delta(u) \ge D(u, v)$) so condition (29) is satisfied. Thus f maximizes $DF(\cdot, \delta(t))$ after line 7 is executed. The values of f and δ remain unchanged until the next execution of line 5. Therefore (by induction) the flow f calculated at line 5 (i.e., f_i) maximizes $DF(\cdot, \delta(t))$ for the $\delta(t)$ before (i.e., $\delta_{i-1}(t)$) and for the $\delta(t)$ after (i.e., $\delta_i(t)$) the next execution of line 7.

Let $P = [[u_0, u_1, \ldots, u_q, u_{q+1}], [d_1, d_2, \ldots, d_q, d_{q+1}]]$, where $u_0 = s$ and $u_{q+1} = t$, be a shortest generalized path from u_0 to u_{q+1} in $N_{f_{i+1}}$. Then:

1. P is a *nzcgp* in $N_{f_{i+1}}$
2. $\delta_{i+1}(u_k) = \sum_{j=0}^{k-1} \widetilde{D}(u_j, u_{j+1}, d_{j+1}),\ \forall\, k,\ 1 \le k \le q+1$ and in particular
3. $\delta_{i+1}(t) = \sum_{j=0}^{q} \widetilde{D}(u_j, u_{j+1}, d_{j+1})$
4. $\delta_{i+1}(u_j) \le \delta_{i+1}(t)\ \forall\, j,\ 0 \le j \le q+1$ (by Lemma 3)

$\forall\, j,\ 0 \le j \le q$, if P's link between u_j and u_{j+1} has non-zero capacity in the direction of P in N_{f_i} then $\delta_i(u_{j+1}) \le \delta_i(u_j) + \widetilde{D}(u_j, u_{j+1}, d_{j+1})$. If the link has zero capacity in the direction of P in N_{f_i} then g_i, where $g_i = f_{i+1} - f_i$, has a positive flow from u_{j+1} to u_j since by item 1 above the link's capacity in the direction of P in $N_{f_{(i+1)}}$ is non-zero. Therefore the link is in N_{f_i,δ_i} and hence $\delta_i(u_{j+1}) = \delta_i(u_j) + \widetilde{D}(u_j, u_{j+1}, d_{j+1})$. Applying the just noted $\delta_i(\cdot)$ inequality and equality along all the nodes of P yields $\delta_i(t) \le \sum_{j=0}^{q} \widetilde{D}(u_j, u_{j+1}, d_{j+1})$ and hence by item 3 above, $\delta_i(t) \le \delta_{i+1}(t)$.

If $\delta_i(t) = \delta_{i+1}(t)$, then $\delta_i(u_{j+1}) = \delta_i(u_j) + \widetilde{D}(u_j, u_{j+1}, d_{j+1})\ \forall j,\ 0 \le j \le q$, hence:

5. $\delta_i(u_{j+1}) - \delta_i(u_j) = \widetilde{D}(u_j, u_{j+1}, d_{j+1})\ \forall\, j,\ 0 \le j \le q$

and $\delta_i(u_k) = \sum_{j=0}^{k-1} \widetilde{D}(u_j, u_{j+1}, d_{j+1}),\ \forall\, k,\ 1 \le k \le q+1$. Hence by item 2 above:

6. $\delta_i(u_j) = \delta_{i+1}(u_j),\ \forall\ j,\ 0 \le j \le q+1$

and then by items 4 and 6 above:

7. $\delta_i(u_j) \le \delta_i(t)\ \forall\ j,\ 0 \le j \le q+1$

By items 5 and 7 above, all nodes and links of P are in N_{f_i,δ_i} (see definition of $N_{f,\delta}$ on page 11). Therefore by item 1 above, after the flow $\hat{g}_i$ is imposed upon N_{f_i,δ_i}, P is a flow augmenting generalized path in N_{f_i,δ_i}. But this is a contradiction since $\hat{g}_i$ is a maximum flow in N_{f_i,δ_i}. Therefore $\delta_{i+1}(t) \neq \delta_i(t)$ and hence $\delta_{i+1}(t) > \delta_i(t)$.

The $\delta_i(t)$ form a strictly ascending sequence, each $\delta_i(t)$, for $i \le i_{max}$, is the delay of a simple *nzcg*p in N (since by Lemma 3, no *nzcndgl*s are in an N_{f_i}) and there are a finite number of simple generalized paths in N. Therefore the number of δ_i(t) is finite and the algorithm will terminate, i.e., $\delta_{i_{\max}+1}(t) = \infty$. Further $\delta_0(t) \ge 0$ since δ_0(t) is the delay of a shortest path. Hence $\delta_i(t) > 0\ \forall\ i,\ 0 < i \le i_{max}$.

Assume that for some $i,\ 0 < i \le i_{max}$, there is a network flow h such that $DF(h, T) > DF(f_i, T)$ for some T, $\delta_{i-1}(t) < T < \delta_i(t)$. Then since $DF(\cdot, T)$ is a linear function of T for a fixed network flow, either $DF(h, \delta_{i-1}(t)) > DF(f_i, \delta_{i-1}(t))$ or $DF(h, \delta_i(t)) > DF(f_i, \delta_i(t))$. This however is a contradiction and therefore f_i provides a maximum $DF(\cdot,T)$ for $\delta_{i-1}(t) \le T \le \delta_i(t)$. Now assume that for some $T > \delta_{i_{\max}}(t)$, there is a network flow h such that $DF(h, T) > DF(f_{i_{\max}+1}, T)$. Since $f_{i_{\max}+1}$ maximizes $DF(\cdot, \delta_{i_{\max}}(t))$, $DF(h, \delta_{i_{\max}}(t)) \le D(f_{i_{\max}+1}, \delta_{i_{\max}}(t))$. Hence $|h| > |f_{i_{\max}+1}|$ and consequently $f_{i_{\max}+1}$ is not a maximum network flow. Since $f_{i_{\max}+1}$ is not a maximum network flow in N, there exists a flow augmenting generalized path from s to t in $N_{f_{i_{\max}+1}}$ and therefore the algorithm will not terminate with $\delta_{i_{\max}}(t)$ as the last value of $\delta(t)$ before $\delta(t) = \infty$. This however is a contradiction, therefore $f_{i_{\max}+1}$, whose magnitude is returned, is a maximum network flow and provides a maximum $DF(\cdot,T)$ for $T \ge \delta_{i_{\max}}(t)$.

$DF(f, T)$ is the message length that can be transferred by a network flow f in time T, $\forall\ T,\ 0 < T < \infty$, and in general $DF(f, T) = DF(f, \overline{T}) + |f|(T - \overline{T})$, $\forall\ T > \overline{T}$. Therefore $DF(f_i, T) = \sigma_{i-1} + |f_i|(T - \delta_{i-1}(t))$, $\forall\ T > \delta_{i-1}(t)$, since in our algorithm, $\sigma_{i-1} = DF(f_i, \delta_{i-1}(t))$. $F(T)$ is the maximum message length that can be transferred from s to t in the time T and $DF(f_i, T)$ is the maximum value of $DF(\cdot, T)$ for $\delta_{i-1}(t) \le T \le \delta_i(t)$ $\forall\ i,\ 1 \le i \le i_{max} + 1$ (where $\delta_{i_{\max}+1}(t) = \infty$). Therefore $\forall\ i,\ 0 < i \le i_{max} + 1$, $F(T) = \sigma_{i-1} + |f_i|(T - \delta_{i-1}(t))$ for $\delta_{i-1}(t) \le T \le \delta_i(t)$, since, as just shown, $DF(f_i, T) = \sigma_{i-1} + |f_i|(T - \delta_{i-1}(t)),\ \forall\ T > \delta_{i-1}$(t). Hence the pairs ($SIGMA[i]$, $DEL[i]$), for $0 \le i \le i_{max}$, and $|f_{i_{\max}+1}|$ define $F(T)$.

By Lemma 1 (on page 7), each f_i, $0 < i \le i_{max} + 1$, can be decomposed into a forward multipath, a backward multipath and a set of loop flows. M_i (the forward multipath of a decomposition of f_i) therefore exists. Since f_i maximizes $DF(f_i, T)$ for $\delta_{i-1} \le T \le \delta_i$, by Lemma 2 on page 8, M_i can transfer a message of length $DF(f_i, T)$ in time T. Hence $MULT[i]$, for $1 \le i \le i_{max} + 1$, is a quickest multipath for messages of length σ when $SIGMA[i\text{-}1] \le \sigma \le SIGMA[i]$.

Since $\exists$ a simple *nzcgp* from s to t in N whose delay is $\delta_{i_{\max}}(t)$, $\delta_{i_{\max}}(t) \le$ the delay of the longest simple generalized path in N from s to t (i.e., $\le$ the maximum delay of simple generalized paths from s to t). Since any simple generalized path in N has no more than n - 1 links, $DEL[i_{max}] < nD_{max}$. ■

5. ALGORITHM IMPLEMENTATION

5.1 GENERAL ASPECTS OF THE IMPLMENTATION

The generic algorithm that we described and proved to be correct in the previous section might be implemented in many different ways, and the complexity of the algorithm will vary from implementation to implementation. We shall describe in this section two closely related implementations **MTMP1** and **MTMP2** that we developed. The term **MTMP** will be used when referring to both implementations. We have provided pseudocode for **MTMP** below.

Algorithm **MTMP**

1. $f \leftarrow 0$; $\sigma \leftarrow 0$; $p\delta \leftarrow 0$; $\hat{\delta}_R \leftarrow 0$; *DEL* ← empty stack; *SIGMA* ← empty stack; *SIGMA* ← empty stack; *MULTI* ← empty stack; Push *NILL* onto *MULTI*;
2. $N_R \leftarrow$ X_ResidualNetwork($N, f, \hat{\delta}_R$)
3. $\hat{\delta}_R \leftarrow$ Min_Path(N_R);
4. **while** $\hat{\delta}_R(t') < \infty$ **do**
5. $\sigma \leftarrow \sigma + |f|(\hat{\delta}_R(t') - p\delta)$; $p\delta \leftarrow \hat{\delta}_R(t')$;
6. $N_{PR} \leftarrow$ Prune($N_R, \hat{\delta}_R$);
7. $\tilde{g} \leftarrow$ Max_Flow(N_{PR});
8. $g \leftarrow$ Reconstitute($N_{PR}, \tilde{g}, N$);
9. **for** each $(u, v) \in E$ **do**
10. $f(u, v) \leftarrow f(u, v) + g(u, v)$
11. $MP \leftarrow$ Decompose_to_Path_Flows(N, f);
12. Push $\hat{\delta}_R(t')$ onto *DEL*; Push σ onto *SIGMA*; Push *MP* onto *MULTI*;
13. $N_R \leftarrow$ X_ResidualNetwork($N, f, \hat{\delta}_R$);
14. $\hat{\delta}_R \leftarrow$ Min_Path(N_R);
15. return(*DEL, SIGMA, MULTI,* $|f|$);

MTMP uses the following set of algorithms:

1. **X_ResidualNetwork** that creates an "expanded residual" network for finding shortest generalized paths
2. **Min_Path** that finds shortest generalized path delays in a network
3. **Prune** that creates a network for flow maximization
4. **Max_Flow** that finds a maximum flow in a network
5. **Reconstitute** that converts a maximum flow in a pruned residual network to a network flow in the residual network from which the pruned network was derived
6. **Decompose_to_Path_Flows** that generates a forward multipath of a decomposition of a network flow

MTMP1 and MTMP2 differ only in the Max_Flow algorithm that they use, with MTMP1 using a particular "pre-flow push" algorithm and MTMP2 using a particular "augmenting path" algorithm.

The first part of Min_Path is the Dijkstra shortest path algorithm that requires all network link delays be non-negative. The Dijkstra and Max_Flow algorithms are sufficiently discussed in the literature and therefore we shall not discuss them in detail here. However in the next subsection we provide pseudocode for the other algorithms listed above on this page. MTMP uses an "expanded residual" network N_R from which shortest

path delays for all nodes in N_f are determined. Hence MTMP refers to a function $\hat{\delta}_R$ relative to a start path node s'' rather than the δ relative to s used in the general algorithm description of the last section and refers to a terminal path node t' rather than the last section's terminal node t. The flow maximization in MTMP is carried out on a "pruned" version of an expanded residual network N_R rather than on $N_{f,\delta}$ and in Reconstitute, the maximum flow is first extended into a network flow in N_R before being transformed into a network flow in N_f.

5.2 DETAILS OF ROUTINES UTILIZED BY MTMP

The algorithm X_ResidualNetwork, shown below utilizes a "node splitting" network transformation, see Ahuja et. al. (Ahuja et. al. 1991), and link delay and capacity redefinition to produce a network $N_R = (G(V_R, E_R), C_R, D_R)$. N_R's links have counter flow capacities of zero and, taking advantage of relationships described by Edmunds and Karp (Edmunds and Karp 1972), non-negative delays. N_R's nodes and links result from the expansion of N that is illustrated in Figure 2 on page17. Each node $u \in V$ is expanded into two "companion" nodes, u' and u''. Each link $(u, v) \in E$ is expanded into two links (u'', v') with capacity $C_R(u'', v') = C(u, v) - f(u, v)$ and delay $D_R(u'', v') = \hat{D}_R(u, v) = \hat{\delta}_R(u'') - \hat{\delta}_R(v') + D(u, v)$ or, if $C_R(u'', v') = 0$, delay $D_R(u'', v') = \infty$, and (v', u'') with capacity $C_R(v', u'') = f(u, v)$ and delay $D_R(v', u'') = -\hat{D}_R(u, v)$ or, if $C_R(v', u'') = 0$, delay $D_R(v', u'') = \infty$. Two additional links (u', u'') and (u'', u') each with an infinite capacity and zero delay are generated for each node $u \in V$. The resulting N_R therefore has $2n$ nodes and $2m + 2n$ links (though links with zero capacity, which might be considered non-existent, are assigned delays of ∞)[17]. All links of N_R are of the form (x'', y') or (x', y''), i.e., there are no links in N_R of the form (x'', y'') or (x', y').

Algorithm **X_ResidualNetwork**(N, f, δ)

1. $S \Leftarrow V$; $V_R \leftarrow \phi$; $E_R \leftarrow \phi$;
2. **while** $S \neq \tilde{\phi}$ **do**
3. $u \leftarrow$ Pop S; $V_R \leftarrow \{u'\} \cup \{u''\} \cup V_R$; $Adj(u'') \leftarrow \phi$; $Adj(u') \leftarrow \phi$;
4. $S \Leftarrow V$;
5. **while** $S \neq \tilde{\phi}$ **do**
6. $u \leftarrow$ Pop S; $S' \Leftarrow Adj(u)$; $E_R = \{(u', u'')\} \cup \{(u'', u')\} \cup E_R$;
7. **while** $S' \neq \tilde{\phi}$ **do**
8. $v \leftarrow$ Pop S'; $E_R \leftarrow \{(u'', v')\} \cup \{(v', u'')\} \cup E_R$;
9. $\hat{D}_R(u, v) \leftarrow \hat{\delta}_R(u'') - \hat{\delta}_R(v') + D(u, v)$;
10. $Adj(u'') \leftarrow \{v'\} \cup Adj(u'')$; $C_R(u'', v') \leftarrow C(u, v) - f(u, v)$; $D_R(u'', v') \leftarrow \hat{D}_R(u, v)$;
11. **if** $C_R(u'', v') = 0$ **then** $D_R(u'', v') \leftarrow \infty$;
12. $Adj(v') \leftarrow \{u''\} \cup Adj(v')$; $C_R(v', u'') \leftarrow f(u, v)$; $D_R(v', u'') \leftarrow -\hat{D}_R(u, v)$;
13. **if** $C_R(v', u'') = 0$ **then** $D_R(v', u'') \leftarrow \infty$;
14. $Adj(u'') \leftarrow \{u'\} \cup Adj(u'')$; $C_R(u'', u') \leftarrow \infty$; $D_R(u'', u') \leftarrow 0$;
15. $Adj(u') \leftarrow \{u''\} \cup Adj(u')$; $C_R(u', u'') \leftarrow \infty$; $D_R(u', u'') \leftarrow 0$;
16. return N_R

---

[17] Dual links in N_R for each link in N lead to a directed network graph that facilitates finding shortest paths and subsequently a maximum flow. Node splitting accommodates networks in which $\exists\ u, v \in V$ such that $(u, v) \in E$ and $(v, u) \in E$.

X_ResidualNetwork assumes adjacency lists exist for $G(V, E)$ (their creation complexity is O(m), i.e., it assumes that $\forall\, u \in V, \exists\, Adj(u) = \{v \in V: (u, v) \in E\}$. S and S' are stacks, $\tilde{\phi}$ is the empty stack and ϕ is the empty set in the algorithm (as they are in all other routines/algorithms listed in this subsection) and V_R and E_R are respectively the set of nodes and links of N_R[18]. The loop of X_ResidualNetwork that creates V_R (line 3) will run n times and, since its complexity is O(1), the complexity of the totality of all iterations of this loop is O(n). The loop that creates E_R and adjacency lists (lines 6-15) will also run n times. However its inner loop (lines 8-13) will pop S' only once for each link in E during the complete set of outer loop iterations. The complexity of this inner loop is also O(1) and the outer loop instructions have complexity O(1), so the complexity of the totality of all the iterations of the outer loop is O(m). Therefore the complete routine has complexity O(m) (since n -1 $\leq m$).

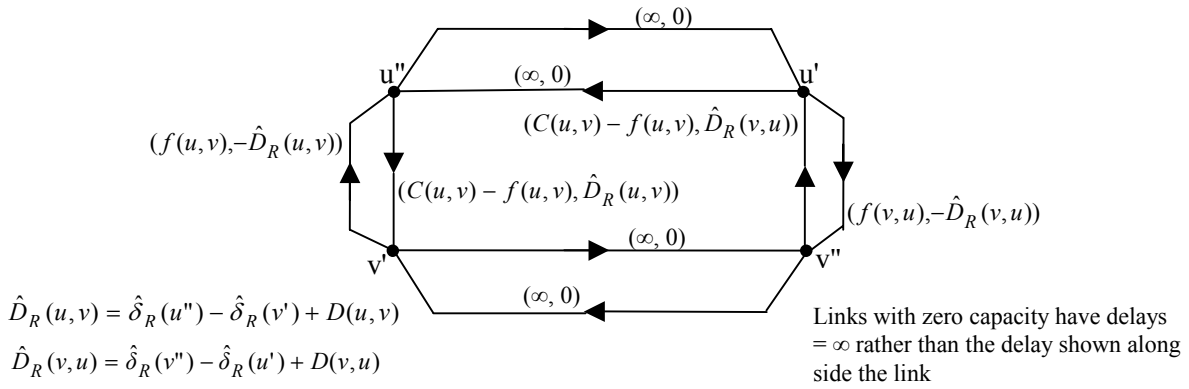

Fig. 2. Creation of expanded residual network.

We shall now show that the link delays of N_R are non-negative. Initially $\forall\, u \in V$, $\hat{\delta}_R(u'') = \hat{\delta}_R(u') = 0$ (line 1 of MTMP on page 15), hence $\hat{D}(u, v) = D(u, v)$. Further, since initially $f = 0$, all links of the form (u', v'') in the initial N_R have infinite delay. Hence the initial N_R's links all have non-negative delay. Subsequent N_R's non-negative delays are a consequence of $\hat{\delta}_R(u'') = \hat{\delta}_R(u') = \delta(u)\ \forall\, u \in V$ at the end of each MTMP line 5 –14 loop iteration. Given this property, equations (27) and (28) on page 12 apply with $\hat{\delta}_R$ substituted for π.

Let $\delta_R(x)$ be the delay of shortest paths from s'' to x in N_R, P_R be a shortest path in N_R, from initial node s'' to terminal node y' and let P_R+ be the one link extension of P_R to y''. Since (y', y'') has a delay of zero, P_R+ has the same path delay as P_R, and therefore $\delta_R(y'') \leq \delta_R(y')$. By an analogous argument $\delta_R(y') \leq \delta_R(y'')$ and hence $\delta_R(y'') = \delta_R(y')$. Min_Path, at the end of each MTMP loop iteration, computes δ_R and determines the new $\hat{\delta}_R$ by setting $\hat{\delta}_R(x)$ to $\hat{\delta}_R(x) + \delta_R(x)$, i.e., it sets the new $\hat{\delta}_R(x)$ to the sum of its old value and the determined $\delta_R(x)$, $\forall\, x \in N_R$. Therefore, since initially $\hat{\delta}_R(y'') = \hat{\delta}_R(y')$ and $\delta_R(y'')$ always equals $\delta_R(y')$, $\hat{\delta}_R(y'')$ always equals $\hat{\delta}_R(y')$.

Corresponding to each link of the form (u'', v') in P_R is a ffl (u, v) and to each link of the form (u', v'') a cfl (v, u) in N_f. This correspondence provides a corresponding generalized path $P(P_R)$ in N_f[19]. Summing link

[18] The symbol $\Leftarrow$, used in this subsection's depictations of X_ResidualNetwork, Prune and Reconstitute, denotes an operation that sets the stack to the left of the symbol to the empty stack and then pushes each element of the set to the right of the symbol onto that stack. Its complexity is O(*number of elements in the set*).

[19] $P(P_R)$ is found by removing each link of the form (u'', u') or (u', u'') from the P_R link sequence and substituting the corresponding N_f links for the remaining links in the sequence.

delays along P_R results in a path delay $\delta_R(y') = \hat{\delta}_R(s'') - \hat{\delta}_R(y') + \sum_{e\in P(P_R)} \tilde{D}(e) = -\hat{\delta}_R(y') + \sum_{e\in P(P_R)} \tilde{D}(e)$, where $\sum_{e\in P(P_R)} \tilde{D}(e)$ is $\tilde{D}(P(P_R))$. Since $\hat{\delta}_R(y')$ is fixed relative to paths from s'' to y' and P_R is a shortest path in N_R, $P(P_R)$ is a shortest generalized path in N_f and hence $\tilde{D}(P(P_R)) = \delta(y)$ where y is the node in N_f from which y' is derived. Hence $\hat{\delta}_R(y') = \delta(y) - \delta_R(y')$ prior to Min_Path's execution. Since Min_Path carries out the operation $\hat{\delta}_R(y') \leftarrow \hat{\delta}_R(y') + \delta_R(y') = \delta(y) - \delta_R(y') + \delta_R(y') = \delta(y)$, after Min_Path's execution $\hat{\delta}_R(y') = \delta(y)$. Consequently equations (27) and (28) hold with $\hat{\delta}_R(u'')$ or $\hat{\delta}_R(u')$ replacing $\pi(u)$ and $\hat{\delta}_R(v'')$ or $\hat{\delta}_R(v')$ replacing $\pi(v)$. Note that in particular $\hat{\delta}_R(t') = \delta(t)$.

The delay of any link of the form *(u", v')* in N_R is $\hat{D}(u, v)$ if $f(u, v) < C(u, v)$. If $\hat{D}(u, v) < 0$, equation (27) requires $f(u, v) = C(u, v)$ and if $f(u, v) = C(u, v)$ link *(u", v')* has infinite delay, hence the delay of *(u", v')* is always non-negative. The delay of any link of the form *(u', v")* in N_R is $-\hat{D}(u, v)$ if $f(u, v) > 0$. If $-\hat{D}(u, v) < 0$, equation (28) requires $f(u, v) = 0$ and if $f(u, v) = 0$ link *(u', v")* has infinite delay. Therefore the delay of *(u', v")* is always non-negative (and if *(u", v')* and *(v', u")* both have finite delay, then $\hat{D}(u, v) = 0$). Thus, since delays of links of the form *(u", u')* and *(u', u")* are zero, for all N_R constructed in MTMP, N_R' s link delays are non-negative.

Since all of N_R's link delays are non-negative, Min_Path uses Dijkstra's algorithm to compute δ_R. Then it computes the new $\hat{\delta}_R$ from δ_R and the old $\hat{\delta}_R$. Min_Path's implementation of the Dijkstra algorithm uses Fibinocchi heaps to facilitate a minimization step which gives the Dijkstra algorithm operating on N_R the complexity $O(2m + 2n + 2n\log 2n) = O(m + n\log n)$. The subsequent computation of the new $\hat{\delta}_R$ has complexity $O(n)$. Hence Min_Path's complexity is $O(m + n\log n)$[20].

Algorithm **Prune**(N_R, $\hat{\delta}_R$)

1. $S \Leftarrow V_R$; $E_{PR} \leftarrow \phi$; $V_{PR} \leftarrow \phi$;
2. **while** $S \neq \tilde{\phi}$ **do**
3. $x \leftarrow$ Pop S;
4. **if** $\hat{\delta}_R(x) \le \hat{\delta}_R(t')$ **then**
5. $V_{PR} \leftarrow \{x\} \cup V_{PR}$; $S' \Leftarrow Adj(x)$; $Adj(x) \leftarrow \phi$;
6. **while** $S' \neq \tilde{\phi}$ **do**
7. $y \leftarrow$ Pop S'
8. **if** $\hat{\delta}_R(y) \le \hat{\delta}_R(t')$, and $\hat{\delta}_R(y) - \hat{\delta}_R(x) = D_R(x, y)$, **then**
9. $E_{PR} \leftarrow \{(x, y)\} \cup E_{PR}$; $Adj(x) \leftarrow \{y\} \cup Adj(x)$;
10. $C_{PR}(x, y) \leftarrow C_R(x, y)$;
11. return(N_{PR})

---

[20] Delays of *(u", v')* and *(u', v")* type N_R links, when not infinite, could be defined to be $D(u, v)$ and $-D(u, v)$ respectively. Then Dijkstra's algorithm couldn't be used since N_R might have negative delay links. However, the Bellman-Ford algorithm, whose complexity is $O(nm)$, could be used since N_R wouldn't have negative delay generalized loops (see Appendix B). This increased complexity versus Dijkstra's algorithm wouldn't, as will be subsequently clear, increase MTMP's complexity.

Prune, shown on page 18, produces a network $N_{PR} = (G(V_{PR},E_{PR}),C_{PR})$ by removing all nodes $x \in V_R$ from N_R for which $\hat{\delta}_R(x) > \hat{\delta}_R(t')$ and removing all links $(x, y) \in E_R$ from N_R for which $\hat{\delta}_R(x) > \hat{\delta}_R(t')$, $\hat{\delta}_R(y) > \hat{\delta}_R(t')$, or $\hat{\delta}_R(y) - \hat{\delta}_R(x) \neq D_R(x, y)$. Note that if links (u, v) and (v, u) are both in E, then a necessary condition for either (u'', v') and/or its corresponding (v', u'') and (v'', u') and/or its corresponding (u', v'') to be in N_{PR} is $D(u, v) = D(v, u) = 0$. For if both link types are present then $\delta(v) - \delta(u) - D(u, v) = \delta(u) - \delta(v) - D(v, u) = 0$ and this can only occur if $D(u, v) = D(v, u) = 0$. Prune, since it requires one computation for each link $(x, y) \in E_R$ (its inner loop, lines 7 - 10, pops S once for each link in E_R during the complete set of outer loop iterations), has complexity $O(2m + 2n) = O(m)$.

Max_Flow returns a flow $\tilde{g}$ in N_{PR}. Max_Flow for MTMP1 is the preflow push algorithm of Cheriyan e.t. al. (Cheriyan e.t. al. 1990) that has complexity $O((2n)^3/\log(2n)) = O(n^3/\log n)$. Max_Flow for MTMP2 is the original Ford-Fulkerson augmenting path algorithm that has complexity $O(m|\tilde{g}|)$, where $\tilde{g}$ is a maximum flow in N_{PR}, see e.g., Ahuja et. al. (Ahuja e.t. al. 1991). Reconstitute, shown below on this page, extends $\tilde{g}$ to a network flow $\hat{g}$ in N_R and then converts $\hat{g}$ to network flow g in N_f. Reconstitute's first and second loops have complexity $O(2n + 2m)$ and its third loop has complexity $O(m)$. Hence Reconsitute's complexity is $O(m)$.

Algorithm **Reconstitute**(N_{PR}, $\tilde{g}$, N)

1. $S \Leftarrow E_R$;
2. **while** $S \neq \tilde{\phi}$ **do**
3. $\quad (x, y) \leftarrow$ Pop S; $\hat{g}(x, y) \leftarrow 0$;
4. $S \Leftarrow E_{PR}$;
5. **while** $S \neq \tilde{\phi}$ **do**
6. $\quad (x, y) \leftarrow$ Pop S; $\hat{g}(x, y) \leftarrow \tilde{g}(x, y)$;
7. $S \Leftarrow E$
8. **while** $S \neq \tilde{\phi}$ **do**
9. $\quad (u, v) \leftarrow$ Pop S; $g(u, v) \leftarrow \hat{g}(u'', v') - \hat{g}(v', u'')$;
10. return g;

It can be shown by straightforward (though tedious) algebraic manipulations that a maximum flow on N_{PR} is transformed by Reconstitute to a maximum flow on N_f. Therefore if Decompose_to_Path_Flows is a valid multipath decomposition, then MTMP is an implementation of the generic algorithm shown on page 11 of Section 4.

Lines 1-4 of Decompose_to_Path_Flows, which is shown on page 20, create a network on which the decomposition will be carried out. They include in the network all links and only those links (u, v) of N for which $f(u, v) > 0$. Here inclusion takes the form of entering v into u's adjacency list[21]. Since f is a network flow between s and t, if (u, v) is a link in the created network and $v \neq t$ then v has at least one node in its adjacency list. The network creation is dominated by its line 4 loop that has complexity $O(m)$.

---

[21] Each $\overline{Adj}(\cdot)$ in Decompose_to_Path_Flows is a linked list and $\hat{\phi}$ at lines 2 and 5 is the empty linked list. Previously we have not explicitly concerned ourselves with ordering of an adjacency list (treating it simply as a set). However adjacency lists are implemented here as linked lists to assure that Decompose_to_Path_Flows' removal of nodes from adjacency lists, at lines 19 and 22, are O(1) complexity operations (which they are since u_i at lines 19 and 22 is the head of the linked list from which it is deleted).

Algorithm **Decompose_to_Path_Flows**(*N, f*)

1. $MP \leftarrow \tilde{\phi}$; $\psi(s) \leftarrow 1$; $\psi(t) \leftarrow 1$; $u_0 \leftarrow s$;
2. $\forall u \in V$, $\overline{Adj}\ (u) \leftarrow \hat{\phi}$;
3. $\forall (u, v) \in E$ **do**
4. **if** $f(u, v) > 0$ **then** Insert v at head of $\overline{Adj}\ (u)$;
5. **while** $\overline{Adj}\ (s) \neq \hat{\phi}$ **do**
6. $\forall u \in \mathrm{V} - \{s, t\}$ **do** $\psi(u) \leftarrow 0$; $\varphi(u) \leftarrow$ NILL;
7. $u_1 \leftarrow$ head of $\overline{Adj}\ (s)$; $\varphi(u_1) \leftarrow s$; $i \leftarrow 1$; $min\_flow \leftarrow f(s, u_1)$;
8. **while** $\psi(u_i) \neq 1$ **do**
9. $u_{i+1} \leftarrow$ head of $\overline{Adj}\ (u_i)$; $\varphi(u_{i+1}) = u_i$; $\psi(u_i) \leftarrow 1$; $i \leftarrow i + 1$;
10. **if** $f(u_{i-1}, u_i) < min\_flow$ **then** $min\_flow \leftarrow f(u_{i-1}, u_i)$;
11. **if** $u_i = t$ **then**
12. $start \leftarrow s$; $\tilde{P} \leftarrow (\Phi(t), min\_flow)$; Push $\tilde{P}$ onto MP;
13. **else**
14. $start \leftarrow u_i$; $min\_flow \leftarrow f(u_{i-1}, u_i)$; $j \leftarrow i$ -1;
15. **while** $u_j \neq start$ **do**
16. **if** $f(u_{j-1}, u_j) < min\_flow$ **then** $min\_flow \leftarrow f(u_{j-1}, u_j)$;
17. $j \leftarrow j$ -1;
18. **while** $u_{i-1} \neq start$ **do**
19. **if** $f(u_{i-1}, u_i) = min\_flow$ **then** Delete $\{u_i\}$ from $\overline{Adj}\ (u_{i-1})$;
20. **else** $f(u_{i-1}, u_i) \leftarrow f(u_{i-1}, u_i) - min\_flow$;
21. $i \leftarrow i$ -1;
22. **if** $f(start, u_i) = min\_flow$ **then** Delete $\{u_i\}$ from $\overline{Adj}\ (start)$;
23. **else** $f(start, u_i) \leftarrow f(start, u_i) - min\_flow$;
24. return(MP);

The main loop of Decompose_to_Path_Flows, lines 6-23, determines, at each iteration, either a simple path that is part of a multipath from s to t or a network loop. It removes a link or links (line 19 and/or line 22) to eliminate the determined path or network loop from the network. This link removal, since it is part of a path or network loop elimination, preserves the property that if (u, v) is a link in the created network and $v \neq t$ then v has at least one node in its adjacency list. The line 5 condition that determines if another iteration will take place is a necessary condition for the existence of a path from s to t in the network. Lines 6 and 7 set up for finding a new path or network loop and then the line 9 - 10 loop finds the path or network loop.

When a path, $\Phi(t) = [s, \ldots, \varphi(\varphi(t)), \varphi(t), t]$, is found, line 12 saves the path $\Phi(t)$ and saves a path flow equal to $\Phi(t)$'s minimum link flow. When a network loop is found, lines 14 - 17 find its minimum link flow. The line 19 - 21 loop and lines 22 - 23 are the path or network loop elimination steps. These steps reduce the link flows of the path or network loop by the minimum path or network loop link flow. The resulting link flows continue to form a network flow since a complete path or network loop flow has been eliminated. The process will

continue until all flow from *s* has been eliminated and when the process terminates, the multipath *MP* will have a flow equal to the initial magnitude of the *s* to *t* network flow.

The main loop is dominated by line 12, and its line 9-10, line 16-17 and line 19-21 interior loops. Each has complexity O(*n*). Hence a single iteration of the main loop, has complexity O(*n*). Since at least one link is eliminated in each iteration, the main loop will be executed no more than *m* times. Therefore the algorithm's complexity is O(*nm*)[22].

5.3 MTMP1 AND MTMP2 COMPLEXITIES

The complexities of lines of the main MTMP algorithm loop that determines the algorithm's complexity are shown in Table 1 below. Adding the table's entries for MTMP1 (taking into account that $n - 1 \leq m < n^2$)) yields a MTMP1 loop complexity of $O((n^3/\log n) + nm)$. If *N*'s link delays are all non-negative integers (even when its link capacities are only restricted to be greater than zero), then each $\hat{\delta}_R(t')$ computed by Min_Path is an integer. Therefore the number of main loop iterations is $\leq nD_{max}$ where, as defined earlier, D_{max} is *N*'s maximum link delay (since, by Section 4's theorem on page 11, the sequence of values of $\hat{\delta}_R(t')$ that is computed by MTMP1 is strictly increasing with the initial value being non-negative and the final value being $< nD_{max}$). Consequently MTMP1's complexity is $O(((n^3/\log n) + nm)nD_{max}) = O(n^2((n^2/\log n) + m)D_{max})$ for non-negative integer link delays.

Table 1. Complexity of routines within MTMP

Loop Line(s)	Complexity	
	MTMP1	MTMP2
$\sigma + \lvert f \rvert(\delta_R(t') - p\delta)$	O(*n*)	O(*n*)
Prune(*NR*, δ)	O(*m*)	O(*m*)
Max_Flow(N_R)	$O(n^3/\log n)$	$O(m\lvert \widetilde{g} \rvert)$
Reconstitute(N_{PR}, $\widetilde{g}$, *N*)	O(*m*)	O(*m*)
$\forall (u, v) \in E, f(u, v) \leftarrow f(u, v) + g(u, v)$	O(*m*)	O(*m*)
Decompose_to_Path_Flows(*N*, *f*)	O(*nm*)	O(*nm*)
X_ResidualNetwork(*N*, *f*)	O(*m*)	O(*m*)
Min_Path(N_R)	$O(m + n\log n)$	$O(m + n\log n)$
The expression for MTMP2's Max_Flow complexity assumes positive integer link capacities.		

If the link capacities are positive integers (even when the link delays are only restricted to be greater than or equal to zero), then the main loop of MTMP will be executed less than nC_{max} times, where, as defined earlier, C_{max} is the maximum link capacity. This is because the positive integer restriction on the link capacities assures that each iteration of the loop adds at least one unit to the network flow, *f*, (i.e., every $\lvert \widetilde{g} \rvert \geq 1$) and because the flow into *t* is limited to be less than or equal to the sum of the capacities of the links into *t* which in turn is less than nC_{max}. Therefore MTMP1's complexity is $O(n^2((n^2/\log n) + m)C_{max})$ for positive integer link capacities and $\underline{O(n^2((n^2/\log n) + m)\min(D_{max}, C_{max}))}$ for non-negative integer link delays and positive integer link capacities.

[22] The use of flow augmenting generalized paths in MTMP2 opens up the possibility that MTMP2's complexity can be improved by combining Max_Flow and Decompose_to_Path_Flows. This combination would create a quickest multipath by adding one generalized path flow at a time to an evolving multipath. Aspects of this approach are discussed in Appendix D Subsections D.3 to D.5.

The complexity of the totality of the executions of MTMP2's Max_Flow routine is $O(nmC_{max})$ when all link capacities are positive integers. This is because the sum of the maximum flows determined by MTMP2's Max_Flow routine (i.e., the sum of all the $\tilde{g}$ computed during MTMP's execution) is less than nC_{max}. Excluding Max_Flow, the complexity of the MTMP2's main loop is $O(nm)$. Since MTMP2's main loop is executed less than nC_{max} times when all link capacities are positive integers, the complexity of the combined executions, excluding MTMP2's Max_Flow is $O(n^2mC_{max})$. Hence MTMP2's complexity is $O(n^2mC_{max})$ when N's link capacities are positive integers (even when its link delays are only restricted to be non-negative). If in addition the link delays are non-negative integers, then MTMP2's complexity is $\underline{O(nm(C_{max} + n\min(D_{max}, C_{max}))}$ (since MTMP2's main loop will not be executed more than nD_{max} times when the link delays are non-negative integers).

The commonly accepted definition of a polynomial complexity for a network algorithm is a complexity that is polynomial in the number of network links and nodes, the log of link delays and the log of link capacities. The commonly accepted definition of a pseudopolynomial complexity is a complexity that isn't a polynomial complexity but is polynomial in the number of links, number of nodes, link delays and link capacities, see e.g., Ahuja et. al. (Ahuja e.t. al. 1991) . Thus by the commonly accepted definitions, MTMP1 and MTMP2 are pseudopolynomial.

6. CONCLUSION

The MTMP algorithms that we developed and discussed above generate a multipath path table for a network where each multipath is a quickest multipath over an applicable interval of message lengths and where the union of these intervals is the positive reals. Hence the algorithms also produce the quickest multipath delays for messages of every possible length. When all network link delays are non-negative integers, the MTMP1 algorithm has a pseudopolynomial complexity, with this complexity being linear in D_{max}, the maximum link delay, and polynomial in the number of nodes and number of links. When all network link capacities are positive integers, the MTMP1 and MTMP2 algorithms have a pseudopolynomial complexity, with this complexity being linear in C_{max}, the maximum link capacity, and polynomial in the number of nodes and number of links.

The message length intervals over which any multipath path table entry except the first and last is a quickest multipath path is a closed interval and the first and last intervals are respectively open below and closed above and closed below and open above. Corresponding to each message length interval is a delay time interval over which any multipath path table entry is a quickest multipath. When the link capacities are all positive integers, the end points are integer message length end points and when the link delays are all non-negative integers, these intervals' end points are integer time end points[23].

Since the algorithm has potential applicability in network message routing systems, extending it to finding quickest multipaths in dynamic cases where link bandwidths have been previously reserved for earlier messages is a natural follow-on to this work. Also, consideration of situations in which segment headers have to be attached to each message segment is a worthwhile extension of this work.

[23] All the characteristics of the algorithms that result from non-negative integer delays and/or positive integer capacities can be routinely extended to situations where the delays are non-negative and/or the capacities are positive rational numbers.

REFERENCES

Ahuja, R. K., T. L. Magnanti, J.B. Orlin 1993. *Network Flows*, Prentice Hall Inc., Upper Saddle River N.J.

Ahuja, R. K., J. B. Orlin 1991. *Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems*, Naval Logistics Research Quarterly, 38, 413-430.

Cheriyan, J, T. Hagerup, and K. Mehlhorn 1990. *Can a maximum flow be computed in O(nm) time?* Proceedings of the 17th International Colloquium on Automata, Languages and Programming 235-248.

Cheriyan, J. and S. N. Maheshwari, 1989. *Analysis of preflow push algorithms for maximum network flow,* SIAM Journal on Computing, 18, 1057-1086.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest 1990. *Introduction to Algorithms*, MIT Press, Cambridge, MA.

Edmunds, J. and R. M. Karp. April 1972. *Theoretical improvements in algorithmic efficiency for network flow problems*. Journal of the Association for Computing Machinery Vol. 19, No. 2.

Ford, L. R. Jr. and D. R. Fulkerson 1962. *Flows in Networks*, Princeton University Press, Princeton, N.J.

Goldberg, A. V. 1985. *A new max-flow algorithm*, Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, MIT, Cambridge MA.

Goldberg, A. V. and R. E. Tarjan 1986. *A new approach to the maximum flow problem,* Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, 136-146.

Kagaris, D., G. E. Pantziou, S. Tragoudas, C. D. Zaroliagis 1999, *Transmissions in a Network with Capacities and Delays*, Networks, 33: 167-174.

Rao, N. S. V. and S. G. Batsell 1997. *Algorithm for minimum end-to-end delay paths,* IEEE Communication Letters, 1(5): 152-154.

Rao, N. S. V. and S. G. Batsell. 1998. *QOS routing via multiple paths using bandwidth reservation*, Oak Ridge National Laboratory report ORNL/TM-1357.

Sleator, D. D. and R. E. Tarjan. 1983. *A data structure for dynamic trees*. Journal of Computer and System Sciences, 24, 362-391.

Xue, G. May 2003. *Optimal multichannel data transmission in computer networks,* Computer Communications, Vol. 26, Issue 7, 759-765.

Xue, G., S. Sun and J. B. Rosen 1998. *Fast data transmission and maximal dynamic flow*, Information Processing Letters, 66:127-132.

APPENDIX A. LINEAR PROGRAMMING AND THE GENERAL MINIMAL COST FLOW AND MINIMUM COST MAXIMUM FLOW PROBLEMS

A.1 LINEAR PROGRAMMING FORMULATION OF THE GENERAL MINIMAL COST FLOW PROBLEM

Ford and Fulkerson (Ford and Fulkerson 1962) formulated their general minimal cost flow problem as follows: Find $f(u, v)$ $\forall (u, v) \in E$ so that

$$|f|T - \sum_{(u,v)\in E} f(u,v)D(u,v) \tag{31}$$

is maximized subject to the constraints

$$\forall u \in V\ u \neq s, t \qquad \sum_{v\in V:(u,v)\in E} f(u,v) \quad - \sum_{v\in V:(v,u)\in E} f(v,u) \ = \ 0 \tag{32}$$

$$\forall (u, v) \in E \quad 0 \leq f(u, v) \leq C(u, v) \tag{33}$$

where $|f|$ is the net flow into t which is equal to the net flow out of s, T is a positive integer, and $\forall (u, v) \in E$, $D(u, v)$ is a non-negative integer and $C(u, v)$ is a positive integer. Note that the definitions of $|f|$ can be written in the form of the constraint equations (32) as follows:

$$\sum_{u\in V:(t,u)\in E} f(t,u) \quad - \sum_{u\in V:(u,t)\in E} f(u,t) + |f| \ = \ 0 \tag{34}$$

$$\sum_{u\in V:(s,u)\in E} f(s,u) \quad - \sum_{u\in V:(u.s)\in E} f(u,s) - |f| \ = \ 0 \tag{35}$$

Let A be an $(m+n)$ x $(m+1)$ matrix, order the links and let $f_1 = f(e_1), f_2 = f(e_2) \ldots, f_m = f(e_m)$ and $f_{m+1} = |f|$. Then we can represent equations (32) to (35) and a requirement that $|f| \geq 0$, by:

$$\begin{array}{c} \text{nodes} \\ \\ \\ \\ \hline \\ \text{links} \end{array} \begin{array}{c} \text{links} \qquad\qquad \vdots\ \text{magnitude} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,m} & a_{1,m+1} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,m} & a_{2,m+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \ldots & a_{n,m} & a_{n,m+1} \\ \hdashline a_{n+1,1} & a_{n+1,2} & \ldots & a_{n+1,m} & a_{n+1,m+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m+n-1,1} & a_{m+n-1,2} & \ldots & a_{m+n-1,m} & a_{m+n-1,m+1} \\ a_{m+n,1} & a_{m+n,2} & \ldots & a_{m+n,m} & a_{m+n,m+1} \end{bmatrix} \end{array} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_m \\ f_{m+1} \end{bmatrix} \begin{array}{c} = \\ \hdashline \leq \end{array} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \hdashline C(e_1) \\ C(e_2) \\ \vdots \\ C(e_{m-1}) \\ C(e_m) \end{bmatrix} \tag{36}$$

$$\text{and } \forall i,\ 1 \leq i \leq m+1 \qquad f_i \geq 0 \tag{37}$$

Here, in equation (36) the ='s sign above and ≤'s sign below the dashed line indicate that in the vector resulting from the left hand side matrix multiplication the first n elements are equal to and the last m elements less than or equal to their right hand side counterparts. The first m columns represent the network links and the last column represents the flow magnitude. The first n rows of the matrix represent the network nodes with the

n -1st and nth row representing respectively nodes t and s. The first n - 2 rows therefore assert that the total inflow to a node equals the total outflow (i.e., equation (32)) and the n - 1st and nth rows define $|f|$ in terms of the flow into and out of respectively t and s (i.e., equations (34) and (35)). The final m rows represent the network links and define the link capacity limitations. Therefore for $1 \le i \le n$, if row i represents node u, and $1 \le j \le m+1$:

$$a_{i,j} = \begin{cases} -1 & e_j = (v, u) \text{ for some } v \in V \text{ or } u = s \text{ (i.e., } i = n) \text{ and } j = m+1 \\ 1 & e_j = (u, v) \text{ for some } v \in V \text{ or } u = t \text{ (i.e., } i = n-1) \text{ and } j = m+1 \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

The last m rows represent the capacity upper bounds and hence for $1 \le i \le m$ and $1 \le j \le m+1$

$$a_{n+i,j} = \begin{cases} 1 & j = i \\ 0 & j \ne i \end{cases} \tag{39}$$

Note that each of the first m columns has two elements equal to 1, one element equal to -1 and all other elements equal to 0 (e.g., if the column represents link (u, v), its element for the row representing u is 1, for the row representing v is -1, and for the row representing (u, v) is 1). The last column has a -1 element and + 1 element in the rows representing nodes s and t respectively and zeroes for all its other elements.

A.2 LINEAR PROGRAM DUALITY

We shall consider general linear program duality, utilizing the equations (36) and (37) type of constraint representation. Here we define a primal problem to be: Find a set of real values $f_1, f_2, \ldots, f_q$ that maximizes

$$\sum_{i=1}^{q} h_i f_i \tag{40}$$

subject to the constraints:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,q-1} & a_{1,q} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,q-1} & a_{2,q} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,q-1} & a_{k,q} \\ a_{k+1,1} & a_{k+1,2} & \cdots & a_{k+1,q-1} & a_{k+1,q} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{p-1,1} & a_{p-1,2} & \cdots & a_{p-1,q-1} & a_{p-1,q} \\ a_{p,1} & a_{p,2} & \cdots & a_{p,q-1} & a_{p,q} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{q-1} \\ f_q \end{bmatrix} \begin{matrix} \\ \\ \\ = \\ \\ \\ \\ \\ \\ \le \\ \\ \\ \end{matrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \\ \hline b_{k+1} \\ b_{k+2} \\ \vdots \\ b_{p-1} \\ b_p \end{bmatrix} \tag{41}$$

$$\text{and } \forall i,\ l+1 \le i \le q \quad f_i \ge 0 \tag{42}$$

The dual of the primal problem is: Find a set of real values $\pi_1, \pi_2, \ldots, \pi_p$ that minimizes

$$\sum_{i=1}^{p} b_i \pi_i \tag{43}$$

subject to the constraints:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,p-1} & a_{1,p} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,p-1} & a_{2,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{l,1} & a_{l,2} & \cdots & a_{l,p-1} & a_{l,p} \\ a_{l+1,1} & a_{l+1,2} & \cdots & a_{l+1,p-1} & a_{l+1,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{q-1,1} & a_{q-1,2} & \cdots & a_{q-1,p-1} & a_{q-1,p} \\ a_{q,1} & a_{q,2} & \cdots & a_{q,,p-1} & a_{q,p} \end{bmatrix} \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_{p-1} \\ \pi_p \end{bmatrix} \begin{matrix} = \\ \\ \\ \\ \\ \geq \\ \\ \\ \\ \end{matrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_l \\ \hline h_{l+1} \\ \vdots \\ h_q \end{bmatrix} \tag{44}$$

$$\text{and } \forall\, i,\ k+1 \leq i \leq p \qquad \pi_i \geq 0 \tag{45}$$

Here we note that the dual's constraint matrix, in equation (44), is the transpose of the primal's constraint matrix, in equation (41).

As we shall see, if a set of values $f_1, f_2, \ldots, f_q$ satisfying equations (41) and (42) and a set of values $\pi_1, \pi_2, \ldots, \pi_p$ satisfying equations (44) and (45) can be found such that:

$$\sum_{i=1}^{q} h_i f_i = \sum_{i=1}^{p} b_i \pi_i$$

then $f_1, f_2, \ldots, f_q$ and $\pi_1, \pi_2, \ldots, \pi_p$ are respectively solutions to the primal and dual problems.

Starting with the primal's function to be maximized, i.e., expression (40), and applying the constraints of the dual's constraint matrix equation (44), to the h_i yields:

$$\sum_{j=1}^{q} h_j f_j = \sum_{j=1}^{l} h_j f_j + \sum_{j=l+1}^{q} h_j f_j \leq \sum_{j=1}^{l} \sum_{i=1}^{p} a_{i,j} \pi_i f_j + \sum_{j=l+1}^{q} \sum_{i=1}^{p} a_{i,j} \pi_i f_j = \sum_{j=1}^{q} \sum_{i=1}^{p} a_{i,j} \pi_i f_j \tag{46}$$

since, by equation (44), $h_j = \sum_{i=1}^{p} a_{i,j} \pi_i$ for $1 \leq j \leq l$ and, by equations (44) and (42) respectively, $h_j \leq \sum_{i=1}^{p} a_{i,j} \pi_i$ and $f_j \geq 0$ for $l+1 \leq j \leq q$.

Starting with the dual's function to be minimized, i.e., expression (43), and applying the constraints of the primal's constraint matrix equation (41), to the b_i yields:

$$\sum_{i=1}^{p} b_i \pi_i = \sum_{i=1}^{k} b_i \pi_i + \sum_{i=k+1}^{p} b_i \pi_i \geq \sum_{i=1}^{k} \sum_{j=1}^{q} a_{i,j} f_j \pi_i + \sum_{i=k+1}^{p} \sum_{j=1}^{q} a_{i,j} f_j \pi_i = \sum_{i=1}^{p} \sum_{j=1}^{q} a_{i,j} f_j \pi_i = \sum_{j=1}^{q} \sum_{i=1}^{p} a_{i,j} \pi_i f_j \tag{47}$$

since by equation (41), $b_i = \sum_{j=1}^{q} a_{i,j} f_j$ for $1 \leq i \leq l$ and by equations (41) and (45) respectively $b_i \geq \sum_{j=1}^{q} a_{i,j} f_j$ and $\pi_i \geq 0$ for $k+1 \leq i \leq p$.

The final expressions in equations (46) and (47) are the same. Hence the equations can be combined to yield the following result: for any $f_1, f_2, \ldots, f_q$ satisfying equations (41) and (42), i.e., the primal constraints, and $\pi_1, \pi_2, \ldots, \pi_p$ satisfying equations (44) and (45), i.e., the dual constraints:

$$\sum_{j=1}^{q} h_j f_j \leq \sum_{j=1}^{q} \sum_{i=1}^{p} a_{i,j} \pi_i f_j \leq \sum_{i=1}^{p} b_i \pi_i \tag{48}$$

Suppose $f_1, f_2, \ldots, f_q$ satisfies the primal constraints, $\pi_1, \pi_2, \ldots, \pi_p$ satisfies the dual constraints and:

$$\sum_{j=1}^{q} h_j f_j \;=\; \sum_{i=1}^{p} b_i \pi_i \tag{49}$$

Then it follows from equations (48) and (49) that for any $f_1^*, f_2^*, \ldots, f_q^*$ satisfying the primal constraints,

$$\sum_{j=1}^{q} h_j f_j^* \;\le\; \sum_{i=1}^{p} b_i \pi_i \;=\; \sum_{j=1}^{q} h_j f_j \tag{50}$$

Hence $f_1, f_2, \ldots, f_q$ maximizes expression (40) and therefore is a solution of the primal problem. Also, for any $\pi_1^*, \pi_2^*, \ldots, \pi_p^*$ satisfying the dual constraints

$$\sum_{i=1}^{p} b_i \pi_i \;=\; \sum_{j=1}^{q} h_j f_j \;\le\; \sum_{i=1}^{p} b_i \pi_i^* \tag{51}$$

Hence $\pi_1, \pi_2, \ldots, \pi_p$ minimizes expression (43) and therefore is a solution of the dual problem.

Equation (49) will hold for a $f_1, f_2, \ldots, f_q$ satisfying the primal constraints and a $\pi_1, \pi_2, \ldots, \pi_p$ satisfying the dual constraints if and only if equality holds in equations (46) and (47) (i.e., $\sum_{j=1}^{q} h_j f_j \;=\; \sum_{j=1}^{q} \sum_{i=1}^{p} a_{i,j} \pi_i f_j$ and $\sum_{i=1}^{p} b_i \pi_i \;=\; \sum_{j=1}^{q} \sum_{i=1}^{p} a_{i,j} \pi_i f_j$). Equality in equation (46), can occur if and only if:

$$\forall\, j,\ l{+}1 \le j \le q \text{ when } h_j < \sum_{i=1}^{p} a_{i,j} \pi_i \text{ then } f_j = 0 \tag{52}$$

(for by constraint (42) $\forall\, j,\ l+1 \le j \le q,\ f_j \ge 0$) and equality in equation (47), can occur if and only if:

$$\forall\, i,\ k{+}1 \le i \le p \text{ when } \pi_i > 0 \text{ then } b_i = \sum_{j=1}^{q} a_{i,j} f_j \tag{53}$$

(for by constraint (41) $\forall\, i,\ k{+}1 \le i \le p,\ b_i \ge \sum_{j=1}^{q} a_{i,j} f_j$).

Thus if for some $f_1, f_2, \ldots, f_q$ satisfying the primal constraints and $\pi_1, \pi_2, \ldots, \pi_p$ satisfying the dual constraints the following hold:

$$\forall\, j,\ l{+}1 \le j \le q \text{ when } h_j < \sum_{i=1}^{p} a_{i,j} \pi_i \text{ then } f_j = 0 \tag{54}$$

$$\forall\, i,\ k{+}1 \le i \le p \text{ when } \pi_i > 0 \text{ then } b_i = \sum_{j=1}^{q} a_{i,j} f_j \tag{55}$$

then $\sum_{j=1}^{q} h_j f_j \;=\; \sum_{i=1}^{p} b_i \pi_i$ and hence $f_1, f_2, \ldots, f_q$ is a solution of the primal problem and $\pi_1, \pi_2, \ldots, \pi_p$ is a solution of the dual problem.

A.3 DUALITY AND THE GENERAL MINIMAL COST FLOW PROBLEM

We note that in the general minimal cost flow problem $l = 0$ (for l of equations (42) and (44) on pages A-2 and A-3) since all of the $f_i \ge 0$. Also $k = n$ (for k of equations (41) and (45) on pages A-2 and A-3) since the inequality applies only to the rows representing network links, $h_i = -D(e_i)\ \forall\, i,\ 1 \le i \le m$, and $h_{m+1} = T$ (for the h_i of expression (40) on page A-2 and equation (44)). The dual constraints are therefore:

$$\begin{array}{c} \text{nodes} \qquad\qquad\qquad\qquad\qquad\qquad \text{links} \\ \left[\begin{array}{cccc:ccc:c} a_{1,1} & a_{2,1} & \cdots & a_{n,1} & a_{n+1,1} & \cdots & a_{n+m-1,1} & a_{n+m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{n,2} & a_{n+1,2} & \cdots & a_{n+m-1,2} & a_{n+m,2} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots \\ a_{1,m} & a_{2,m} & \cdots & a_{n,m} & a_{n+1,m} & \cdots & a_{n+m-1,m} & a_{n+m,m} \\ a_{1,m+1} & a_{2,m+1} & \cdots & a_{n,m+1} & a_{n+1,m+1} & \cdots & a_{n+m-1,m+1} & a_{n+m,m+1} \end{array}\right] \left[\begin{array}{c} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_n \\ \pi_{n+1} \\ \vdots \\ \pi_{n+m} \end{array}\right] \geq \left[\begin{array}{c} -D(e_1) \\ -D(e_2) \\ \vdots \\ -D(e_m) \\ T \end{array}\right] \end{array} \tag{56}$$

$$\forall\, i,\ n+1 \leq i \leq m+n \quad \pi_i \geq 0 \tag{57}$$

We can see from the dual constraint matrix that the first n dual variables are associated with network nodes and the last m dual variables are associated with network links. Further we can see from the constraint equation (57) that the dual variables corresponding to the links are constrained to be non-negative. Since each row of the dual constraint network is a column of the primal constraint network, the first m rows will have only three non-zero elements and the last row only two non-zero elements (see equations (38) and (39) on page A-2). Designating the dual variables by the nodes or links they are associated with, we then have equation (58) from the first m rows of the dual constraint matrix equation, i.e., equation (56), and equation (59) from the last row of the dual constraint matrix equation.

$$\pi(u) - \pi(v) + \pi((u, v)) \geq -D(u, v) \quad \forall\, (u, v) \in E \tag{58}$$

$$\pi(t) - \pi(s) \geq T \tag{59}$$

We can assure that the inequalities of equations (58) and (59) as well as (57), i.e., $\forall\, (u, v) \in E,\ \pi((u, v)) \geq 0$, are met by defining the $\pi((u, v))$, $\pi(s)$ and $\pi(t)$ as follows:

$$\pi((u, v)) = \max(0,\ \pi(v) - \pi(u) - D(u, v)) \tag{60}$$

$$\pi(s) = 0 \quad \text{and} \quad \pi(t) = T \tag{61}$$

We shall now translate conditions (54) and (55) of the general dual problem (on page A-4) to the general minimal cost flow problem. Condition (54) (given the only three nonzero $a_{i,j}$ for $0 \leq j \leq m$ and the only two nonzero $a_{i,j}$ for $j = m + 1$ (see equations (38) and (39)) is:

$$\forall\, (u, v) \in E, \quad \pi(u) - \pi(v) + \pi((u, v)) > -D(u, v) \ \Rightarrow\ f(u, v) = 0 \tag{62}$$

$$\pi(t) - \pi(s) > T \ \Rightarrow\ |f| = 0 \tag{63}$$

Because of the $\pi(t)$ and $\pi(s)$ definitions in equation (61), $\pi(t) - \pi(s) = T$, so we needn't concern ourselves with condition (63). It can be seen from the definition of $\pi((u, v))$ in equation (60) that $\pi(u) - \pi(v) + \pi((u, v))$ will be greater than $-D(u, v)$ if and only if $\pi(v) - \pi(u) < D(u, v)$[24]. Hence the general dual condition (54) translated to the general minimal cost flow problem with the dual variable values of equations (60) and (61) is:

$$\forall\, (u, v) \in E, \quad \pi(v) - \pi(u) < D(u, v) \ \Rightarrow\ f(u, v) = 0 \tag{64}$$

The b_{n+j} of the general dual problem for $1 \leq j \leq m$ can be seen from equation (36) on page A-1 to equal $C(e_j)$ for the general minimal cost flow problem. Hence condition (55) of the general dual problem

[24] If $\pi(v) - \pi(u) \geq D(u, v)$, then $\pi(v) - \pi(u) - D(u, v) \geq 0$. Therefore, by equation (60), $\pi((u, v)) = \pi(v) - \pi(u) - D(u, v)$ and hence $\pi(u) - \pi(v) + \pi((u, v)) = -D(u, v)$. If $\pi(v) - \pi(u) < D(u, v)$, then $\pi(v) - \pi(u) - D(u, v) < 0$. Therefore $\pi((u, v)) = 0$ and hence $\pi(u) - \pi(v) + \pi((u, v)) = \pi(u) - \pi(v) > -D(u, v)$.

translated to the general minimal cost flow problem (given that $a_{n+j,\,j} = 1$ and $a_{n+i,\,j} = 0$ for $i \neq j$, see equation (39)) is:

$$\forall\ (u,\ v) \in E,\quad \pi((u,\ v)) > 0 \ \Rightarrow\ f(u,\ v)\ = C(u,\ v) \tag{65}$$

It can be seen from the definition of $\pi((u,\ v))$ in equation (60) that $\pi((u,\ v))$ will be greater than 0 if and only if $\pi(v) - \pi(u) > D(u,\ v)$[25]. Hence the general dual condition (55) translated to the general minimal cost flow problem with the dual variable relations of (60) is:

$$\forall\ (u,\ v) \in E,\quad \pi(v) - \pi(u) > D(u,\ v)\ \Rightarrow\ f(u,\ v)\ = C(u,\ v) \tag{66}$$

Therefore a procedure that creates link flows and node related dual variables that satisfy the constraints of equations (36) and (37), the equalities of equation (61) and the conditions (64) and (66), will provide a solution to the general minimal cost flow problem. Here equation (61) and conditions (64) and (66) are respectively equation (26) and conditions (28) and (27) of Section 4 of the body of the report (see page 12) that we used to prove the correctness of the general MTMP approach.

A.4 FORD-FULKERSON GENERAL MINIMAL COST FLOW ALGORITHM

The general Ford-Fulkerson General Minimal Cost Flow Algorithm, which we have called **FFGMCF**, is shown below on this page. We have constructed the pseudocode from a verbal description given by Ford and Fulkerson (Ford and Fulkerson 1962) and included the added condition that $\pi(t) < nD_{max}$ which is consistent with an observation made in that reference following the description. Here a node in a residual network N_f is reachable from s if there is a non-zero capacity "admissible" generalized path from s to the node. A generalized path is admissible if each of its links are admissible and a link $(u,\ v)$ is admissible if $\pi(v) - \pi(u) = D(u,\ v)$. $\overline{R}$ in line 10 is the complement of R, so the line 10 operation increments the dual variable associated with each node that isn't in the reachable set.

Algorithm **FFGMCF**$(N,\ f)$

1. $\forall\ (u,\ v) \in E,\ f(u,\ v) \leftarrow 0;\ \ \forall\ u \in V,\ \pi(u) \leftarrow 0;$
2. **while** $\pi(t) \leq T$ and $\pi(t) < nD_{max}$ **do**
3. $\quad R \leftarrow$ reachable set of V;
4. $\quad$ **while** $t \in R$ **do**
5. $\quad\quad P \leftarrow$ a non-zero capacity generalized admissible path from s to t;
6. $\quad\quad \forall\ (u,\ v)$ in P **do**
7. $\quad\quad\quad$ **if** $(u,\ v)$ a ffl in P **then** $f(u,\ v) \leftarrow f(u,\ v)\ +\ \widetilde{C}(P)$;
8. $\quad\quad\quad$ **else** $f(u,\ v)\ \leftarrow f(u,\ v)\ -\ \widetilde{C}(P)$;
9. $\quad\quad R \leftarrow$ reachable set of V;
10. $\quad \forall\ u \in \overline{R}\ ,\ \pi(u) \leftarrow\ \pi(u) + 1;$
11. return(f);

FFGMCF's line 5-9 loop utilizes flow augmenting generalized paths to maximize the flow in a network whose links are the $(u,\ v) \in E$ for which $\pi(v) - \pi(u) = D(u,\ v)$ and whose nodes are the end nodes of such links as called for in the Ford Fulkerson reference. The network is essentially $N_{f,\delta}$ defined in Section 4 on page 11 (where $\pi(t) = \delta(t)$). MTMP finds $\delta(t)$ "directly" by applying a standard shortest path algorithm to a residual network. FFGMCF successively increases $\pi(t)$, line 10, until t is in the reachable set of the residual network (t's entry into the reachable set is termed "breakthrough" in the reference). The result is that for a residual network, $\forall\ u \in V, \pi\ (u) = \delta(u)$ for u such that $\delta(u) \leq \delta(t)$ and $\pi(u) = \delta(t)$ for u such that $\delta(u) \geq \delta(t)$. FFGMCF's

[25] From equation (60), $\pi((u,\ v)) > 0$ if and only if $\pi(v) - \pi(u) - D(u,\ v) > 0$, i.e., if and only if $\pi(v) - \pi(u) > D(u,\ v)$.

correctness is proven in the reference by applying equation (61) and conditions (64) and (66) of this appendix (corresponding to equation (26), and conditions (28) and (27) of Section 4 of the body of the report - see page 12). No consideration was given in the reference to the complexity of the algorithm.

We shall assume in our discussion of FFGMCF's complexity that link capacities are positive integers and link delays are non-negative integers, as is done in the reference. The reachable set R (lines 3 and 9) is determined by a "labeling" method[26] that has complexity O(m). This labeling method includes the maintenance of predecessor nodes. Hence the generalized path P (line 5) is generated from the method's results in O(n). Therefore in the line 5-9 loop, line 9 dominates the rest of the loop from a complexity standpoint. The complexity of the totality of loop iterations executed during FFGMCF is therefore O(nmC_{max}) since the number of augmenting generalized paths is no greater than the maximum network flow which in turn is less than nC_{max}. Line 3 and line 10 are the only main loop instructions not part of (or related to) the inner line 5 - 9 loop and line 3's generation of a new reachable set, that has complexity O(m), dominates line 10, that has complexity O(n). Since no more than nD_{max} iterations of the outer loop will occur, the complexity of the totality of outer loop iterations exclusive of the inner loop contributions is O(nmD_{max}). Hence FFGMCF's complexity is O($nm(C_{max}+ D_{max})$).

Algorithm **CGMCF**(N, f)

1. $\forall$ *(u, v)* $\in E$, $f(u, v) \leftarrow 0$; $\forall u \in V$, $\pi(u) \leftarrow 0$; $T \leftarrow 0$;
2. **while** $T < nD_{max}$ **do**
3. $R \leftarrow$ reachable set of V; *enable* $\leftarrow 0$;
4. **while** $t \in R$ **do**
5. $P \leftarrow$ a non-zero capacity generalized admissible path from s to t;
6. $\forall$ *(u, v)* in P **do**
7. **if** *(u, v)* a ffl in P **then** $f(u, v) \leftarrow f(u, v) + \widetilde{C}(P)$;
8. **else** $f(u, v) \leftarrow f(u, v) - \widetilde{C}(P)$;
9. $R \leftarrow$ reachable set of V; *enable* $\leftarrow 1$;
9a. **if** *enable* = 1 **then** Save(f and T);
10. $\forall u \in \overline{R}$, $\pi(u) \leftarrow \pi(u) + 1$;
10a. $T \leftarrow T + 1$;
11. return(saved (f, T) pairs);

Ford and Fulkerson explicitly recognized that their general minimal cost flow problem defines a set of related problems, one for each of the set of successive integers 0, 1, 2, ..., that for some positive integer T_{max} the solution is a maximum flow for the network[27], that the problem for any integer $\geq T_{max}$ has this maximum flow as a solution and that $T_{max} \leq$ the network's longest generalized path delay. They further recognized that if breakthrough occurs at $\pi(t) = T_1$ and subsequently not again until $\pi(t) = T_2$, then the flow resulting from the maximization immediately after $\pi(t)$ is set to T_1 is a solution for any T between and including T_1 and T_2.

[26] The labeling method first "labels" nodes and subsequently "scans" them. Scanning a node is the process by which unlabeled nodes that can be "reached" via a single link from the node being scanned are labeled. The label given a node contains a predecessor node and a generalized path capacity. The predecessor node is the node which when scanned led to the labeled node being labeled, and the generalized path capacity is the capacity of the generalized path that can be constructed by tracing back from the labeled node to s via predecessor nodes. The process starts with s being labeled with itself as its predecessor node and with a generalized path capacity of ∞. It continues by scanning in no particular order labeled but unscanned nodes and it terminates when no such node exists. Here a node v can be reached from a node u via a single link if and only if either *(u, v)* $\in E$, $f(u, v) < C(u, v)$ and $\pi(v) - \pi(u) = D(u, v)$ or *(v, u)* $\in E$, $f(v, u) > 0$ and $\pi(u) - \pi(v) = D(v, u)$. Note that when $\pi(u) = \delta(u)$ and $\pi(v) = \delta(v)$, the admissibility condition $\pi(v) - \pi(u) = D(u, v)$ is a necessary condition for *(u, v)* to be a link on a shortest generalized path in N_f from s to t.

[27] They noted that such a maximum flow would be a solution to the minimum cost maximum flow problem that is described in the next subsection.

Suppose FFGMCF is altered so T is initialized to 0, T is incremented by one at the end of each iteration of the routine's main loop, and the loop is terminated when $T = nD_{max}$. Further suppose the altered FFGMCF saves the flow and T values on each exit from the line 5-9 loop when at least one augmentation was made during the loop execution. Then the altered FFGMCF, displayed on page A-7 as **CGMCF**, solves the complete set of problems and has the same complexity as FFGMCF[28].

Algorithm **FFMTMP**(*N, f*)

1. $\forall$ *(u, v)* $\in E$, $f(u, v) \leftarrow 0$; $\forall\, u \in V$, $\pi(u) \leftarrow 0$; $T \leftarrow 0$;
2. **while** $T < nD_{max}$ **do**
3. $R \leftarrow$ reachable set of V; *enable* $\leftarrow 0$;
4. **while** $t \in R$ **do**
5. $P \leftarrow$ a non-zero capacity generalized admissible path from s to t;
6. $\forall$ *(u, v)* in P **do**
7. **if** *(u, v)* a ffl in P **then** $f(u, v) \leftarrow f(u, v) + \widetilde{C}(P)$;
8. **else** $f(u, v) \leftarrow f(u, v) - \widetilde{C}(P)$;
9. $R \leftarrow$ reachable set of V; *enable* $\leftarrow 1$;

9a1. **if** *enable* = 1 **then**

9a2. $MP \leftarrow$ Decompose_to_Path_Flows(*N, f*);

9a3. $\sigma \leftarrow$ message length MP can communicate in time T;

9a4. Save(*MP, T,* σ);

10. $\forall\, u \in \overline{R}$, $\pi(u) \leftarrow \pi(u) + 1$;

10a. $T \leftarrow T + 1$;

11. return(saved (*MP, T,* σ) triples);

Further suppose Decompose_to_Path_Flows in Subsection 5.2 on page 20 is executed on exit from CGMCF's line 5-9 loop when at least one augmentation was made during the loop execution, and T, the multipath and the multipath's message length for T are saved. Then the altered CGMCF, shown above on this page as **FFMTMP**, creates a path table of minimum end-to-end delay multipaths. The number of required decompositions is bounded above by the maximum flow which is $< nC_{max}$ and by the number of distinct values of T which is nD_{max}. Therefore the number of decompositions is bounded above by $n\min(C_{max}, D_{max})$ and since such a decomposition has a complexity of $O(nm)$, the complexity of FFMTMP is $O(nm(C_{max} + D_{max}) + n^2 m\min(C_{max}, D_{max})) = O(nm(C_{max} + D_{max} + n\min(C_{max}, D_{max}))$.

A.5 THE MINIMUM COST MAXIMUM FLOW PROBLEM

The Minimum Cost Maximum Flow Problem is the problem of finding a maximum flow f^* in a network $N(G(V,F), C, D)$ such that:

$$\forall \quad f\text{: } f \text{ is a maximum flow,}$$

$$\sum_{(u,v)\in E} f^*(u,v)D(u,v) \le \sum_{(u,v)\in E} f(u,v)D(u,v) \tag{67}$$

[28] $\pi(t) = T$ until the end of each main loop iteration since $\pi(t)$ and T are initialized to 0 and each incrementing of $\pi(t)$ is followed by an incrementing of T (lines 10 and 10a) restoring the equality of $\pi(t)$ and T. Therefore no test for $\pi(t) \le T$ is required at line 2.

Edmunds and Karp (Edmunds and Karp 1972)[29] presented an algorithmic solution to the problem that we provide in pseudocode below on this page as **EKMCMF**. They confined their consideration to networks in which if $(u, v) \in E$ then $(v, u) \notin E$, noting that any network can be transformed into a network meeting this criterion through artifacts such as fictitious nodes.

Algorithm **EKMCMF**(N, f)

1. $\forall\ (u, v) \in E,\ f(u, v) \leftarrow 0;\ \forall\ u \in V,\ \pi(u) \leftarrow 0;\ |f_{max}| \leftarrow$ magnitude of a maximum flow from s to t in N;
2. **while** $|f| < |f_{max}|$ **do**
3. $\quad \forall\ (u, v) \in E^f,\ \hat{D}(u, v) \leftarrow \pi(u) - \pi(v) + \widetilde{D}(u, v)$;
4. $\quad \forall\ u \in V,\ \hat{\delta}(u) \leftarrow$ shortest s to u path delay in $\hat{N}^f$;
5. $\quad P \leftarrow$ shortest path from s to t in $\hat{N}^f$ with minimum number of links among all shortest paths from s to t in $\hat{N}^f$;
6. $\quad \forall\ (u, v)$ in P **do**
7. $\quad\quad$ **if** (u, v) a ffl in P **then** $f(u, v) \leftarrow f(u, v) + \widetilde{C}(P)$;
8. $\quad\quad$ **else** $f(u, v) \leftarrow f(u, v) - \widetilde{C}(P)$;
9. $\quad \forall\ u \in V$ **do** $\pi(u) \leftarrow \pi(u) + \hat{\delta}(u)$;
10. return(f);

$\hat{N}^f = (G(V, E^f,)\ C^f, \hat{D})$ (line 4) is related but not identical to N_f which we defined in the body of this report. $(u, v) \in E^f$ if and only if $(u, v) \in E$ and $f(u, v) < C(u, v)$ or $(v, u) \in E$ and $f(v, u) > 0$. Thus one or more links in E may not be in E^f and E^f may contain one or more links that aren't in E. Analogous to definitions for N_f, $C^f(u, v) = C(u, v) - f(u, v)$ and $\widetilde{D}(u, v) = D(u, v)$ if $(u, v) \in E$, and $C^f(u, v) = f(v, u)$ and $\widetilde{D}(u, v) = -D(v, u)$ if $(u, v) \notin E$. Here $\widetilde{D}(u, v)$ is used at line 3 to compute $\hat{D}(u, v)$ and, with these definitions, $\forall\ (u, v) \in E^f, \hat{D}(u, v) \geq 0$. Further if $(u, v) \in E^f$ and $(v, u) \in E^f$, then $\hat{D}(u, v) = \hat{D}(v, u) = 0$. Though not directly evident from EKMCMF, $\pi(u)$ calculated at line 9 is $\delta(u)$, the minimum s to u generalized path delay in N^f, and consequently EKMCMF anticipated MTMP's use of $\delta(u)$ as a dual variable[30].

We note that Edmunds and Karp did not completely address the problem of the complexity of their algorithm. They did however address bounds on the number of augmenting paths it uses (which equals the number of executions of line 5). Their algorithm (i.e., EKMCMF) which redefines $\hat{N}^f$ prior to each flow augmentation, masks the fact that it is engaging in a series of network flow maximizations of networks defined by successively generated differing values of $\delta(t)$. However they recognized this fact and used it to develop their bound on the number of flow augmentations.

---

[29] This paper is most recognized for a network maximum flow algorithm that appears toward its beginning and which has become known as the Edmunds-Karp algorithm. This algorithm utilizes a modified labeling method which leads to augmenting generalized paths being chosen in a sequence with monotonically increasing numbers of links. It is proven in the paper that the number of augmenting generalized paths is less than n^3 though the particular proof also can be used to establish that this number is less than nm. The procedure outlined for obtaining an augmenting generalized path has complexity $O(m)$, hence the algorithm has complexity $O(nm^2)$. EKMCMF is subsequently outlined in the paper as a lead up to a minimum cost maximum flow algorithm that isn't closely related to an MTMP but has lower complexity than EKMCMF.

[30] The equality is noted in the reference (Edmunds and Karp 1972) in the statement of a theorem for which no proof is provided (and a shortest generalized path delay is incorrectly equated to $\pi^k(u)$ rather than $\pi^{k+1}(u)$).

Edmunds and Karp also addressed the number of operations required to find shortest path delays and as a result presented a routine for determining shortest path delays in a network with non-negative link delays that is used in the line 4 determination of the $\hat{\delta}(\cdot)$. This routine is essentially Dijsktra's algorithm without predecessor nodes and is applicable due to Edmunds and Karp's definition of link delays in $\hat{N}^f$ (i.e., the $\hat{D}(\cdot,\cdot)$) that assures that they will be non-negative. Since, with predecessor nodes, the shortest path routine will not necessarily provide a minimum link shortest generalized path from *s* to *t*, the generalized path *P* of line 5 is generated by their modified "labeling method"[31]. Line 4 whose complexity is O($m + n\log n$) dominates the line 3 to 9 loop (in which line 5 has complexity O(m)). The number of iterations of this loop is bounded above by nC_{max} and n^2mD_{max} when the link capacities are positive integers and the link delays are non-negative integers. The first bound results from the facts that each iteration adds at least one unit to the flow and the maximum flow is bounded above by nC_{max}. The second bound results from the facts that each set of iterations for a particular $\delta(t)$ value corresponds to a flow maximization of a subnetwork of a network $N^{\bar{f}}$, $\delta(t)$ takes on no more than nD_{max} values, and a flow maximization using the modified labeling method requires no more than nm flow augmenting generalized paths. Here $\bar{f}$ is the flow immediately before the first augmenting path is chosen with end-to-end delay equal to the particular $\delta(t)$. Therefore the complexity of EKMCMF is O($n(m + n\log n)\min(C_{max}, nmD_{max})$).

Algorithm **EKMTMP**(*N, f*)

1. $\forall$ *(u, v)* $\in E$, $f(u, v) \leftarrow 0$; $\forall\, u \in V$, $\pi(u) \leftarrow 0$; $|f_{max}| \leftarrow$ magnitude of a maximum flow from *s* to *t* in *N*;
1a. *enable* $\leftarrow 0$;
2. **while** $|f| < |f_{max}|$ **do**
3. $\forall$ *(u, v)* $\in E^f$, $\hat{D}(u, v) \leftarrow \pi(u) - \pi(v) + \tilde{D}(u, v)$;
4. $\forall\, u \in V$, $\hat{\delta}(u) \leftarrow$ shortest *s* to *u* path delay in $\hat{N}^f$;
4a. **if** $\hat{\delta}(t) \neq 0$ and *enable* $\neq 0$ **then**
4b. *MP* $\leftarrow$ Decompose_to_Path_Flows(*N, f*);
4c. $\sigma \leftarrow$ message length *MP* can communicate in time T; Save(*MP, T,* σ);
5. $P \leftarrow$ shortest path from *s* to *t* in $\hat{N}^f$ with minimum number of links among all shortest paths from *s* to *t* in $\hat{N}^f$;
6. $\forall$ *(u, v)* in *P* **do**
7. **if** *(u, v)* a ffl in *P* **then** $f(u, v) \leftarrow f(u, v) + \tilde{C}(P)$;
8. **else** $f(u, v) \leftarrow f(u, v) - \tilde{C}(P)$;
8a. $T \leftarrow \pi(t) + \hat{\delta}(t)$; *enable* $\leftarrow 1$;
9. $\forall\, u \in V$, $\pi(u) \leftarrow \pi(u) + \hat{\delta}(u)$;
9a. **if** *enable* $= 1$ **then**
9b. *MP* $\leftarrow$ Decompose_to_Path_Flows(*N, f*);
9c. $\sigma \leftarrow$ message length MP can communicate in time *T*; Save(*MP, T,* σ);
10. return(*f*);

EKMCMF solves the whole family of generalized minimal cost flow problems formulated by Ford and Fulkerson (Ford and Fulkerson 1962) though it doesn't pause to save the solutions whose flow magnitudes are

[31] This modified labeling method follows the form of the original labeling method of Ford and Fulkerson but scans nodes in the order that the nodes were labeled. It therefore has the form of a breadth first search via, in EKMCMF, links satisfying $\hat{\delta}(v) - \hat{\delta}(u) = \pi(u) - \pi(v) + \tilde{D}(u, v)$, analogous to admissible path links of FFGMCF. Since $\pi(u) - \pi(v) + \tilde{D}(u, v) = \hat{D}(u, v)$, these links satisfy a necessary condition for being links of a shortest path from *s* to *t* in N^f.

less than the maximum network flow. Consequently it can readily be expanded to produce a path table of minimum end-to-end delay multipaths. Such an expansion is presented on page A-10 as **EKMTMP**. It requires detecting a change in $\delta(t)$ (other than its change from its initial value) and after such a change decomposing the flow into a multipath flow. This is accomplished by testing for $\hat{\delta}(t) > 0$ at line 4a after the generation of the $\hat{\delta}(\cdot)$ at line 4 (and by testing a flag which indicates whether or not the current iteration of the line 3-9 loop is its initial iteration). A final decomposition (line 9b) is required after the last loop iteration to handle the maximum network flow (which is a solution to the minimum cost maximum flow problem). Each decomposition has complexity O(nm). The number of required decompositions is bounded above by the maximum network flow and by the maximum number of values of $\delta(t)$. Hence the total set of decompositions has complexity O($n^2m\min(C_{max}, D_{max})$). Adding this to the complexity of EKMCMF yields an EKMTMP complexity of O($n((m + n\log n)\min(C_{max}, nmD_{max}) + nm\min(C_{max}, D_{max}))$).

A.6 COMPARISONS

The most obvious difference between MTMP and the algorithms of Ford and Fulkerson and Edmunds and Karp, translated into FFGMCF and EKMCMF respectively (in sections A.4 and A.5), is MTMP's decomposition of flows into multipaths. Since communications networks were not the focus of Ford and Fulkerson and Edmunds and Karp such decompositions were not of concern in these works[32]. However, as illustrated in the previous two subsections, FFGMCF and EKMCMF can be expanded to provide results comparable to MTMP's. MTMP has another significant difference. Its construction of the set of pruned residual networks allows instantiations of MTMP to use any network flow maximization routine that is applicable to networks that can be represented as directed graphs having non-negative link capacities. Hence MTMP1 uses the flow maximization routine with the apparently "best" complexity of flow maximization routine complexities expressed only in terms of n and m, i.e., independent of link capacities and delays (see Appendix B, page B-2). FFGMCF and EKMCMF are in contrast oriented to network flow maximization via flow augmenting generalized paths and particularly to such maximizations that are achieved through use of a "labeling" technique.

The concept of solving all MTMPP's for a network via generation of a multipath path table through a series of flow maximizations (as is done in MTMP) was anticipated by FFGMCF. Further the use of residual networks' minimum generalized path delays to determine successive networks for flow maximization was anticipated by EKMCMF. However the explicit use of these minimum generalized path delays as dual variables was apparently first implemented in our MTMP. It is worth noting that in the Ford Fulkerson reference it is recognized that FFGMCF's dual variable values for unreachable set nodes, those that comprise $\overline{R}$, can be increased by more than one unit at a time. The increase that is suggested for FFGMCF is the minimum increase necessary to make a node in $\overline{R}$ reachable, i.e., to move the node from $\overline{R}$ to R. It is pointed out there that a maximum of $|\overline{R}_0|$ such increases will then lead to breakthrough, where $\overline{R}_0$ is the complement of the initial reachable set or of the reachable set determined immediately after a flow maximization[33]. However this procedure still requires consideration of $\pi(t)$ values that are between breakthrough values.

Table 2 on page A-12 provides complexities of instantiations of MTMP and expansions of FFGMCF and EKMCMF to produce MTMP results. The column 2-4 entries should be understood as the x values of O(x).

---

[32] It is worth noting however that a flow decomposition algorithm was outlined in the Ford and Fulkerson reference outside the context of the general minimal cost flow problem.

[33] The increment for the $\pi(u)$ for $u \in \overline{R}_0$ is determined here by examining all non-zero capacity links between nodes in R_0 and $\overline{R}_0$ and determining the minimum increment which makes one such link admissible. This can be accomplished with complexity O(m). Since $|\overline{R}_0| < n$, less than n such increments are required to reach breakthrough.

We have added four routines that are mild modifications of other routines listed in the table. MTMP3, MTMP4 and MTMP5 are MTMP instantiations[34] that use different flow augmenting generalized path maximum flow routines than MTMP2 does. MTMP3 uses the modified labeling method of Edmunds and Karp (Edmunds and Karp 1972). MTMP5 uses the apparently best flow augmenting generalized path maximum flow routine from a complexity point of view when complexity is limited to being expressed independent of C_{max} and D_{max}. The complexity of this best routine is O($nm\log n$), see Sleator and Tarjan (Sleator and Tarjan 1993). FMTMPR is FFMTMP except that the modified labeling method of Edmunds and Karp is substituted for the original labeling method of Ford and Fulkerson. Columns 2 - 4 contain algorithm complexities that respectively are those expressed in terms of both C_{max} and D_{max}, D_{max} only, and C_{max} only. A column 3 or 4 entry of "-----" indicates an inability to express the complexity in the column's required terms. The complexities in Table 2 are all applicable to the case where link capacities are positive integers and link delays are non-negative integers. However column 3's and column 4's complexities are also valid respectively in cases where only the link delays are integers and only the link capacities are integers.

Table 2. Algorithm complexities

algorithm	full	independent of C_{max}	independent of D_{max}
MTMP1	$n^2(n^2/\log n + m)\min(C_{max}, D_{max})$	$n^2(n^2/\log n + m)D_{max}$	$n^2(n^2/\log n + m)C_{max}$
MTMP2	$nm(C_{max} + n\min(C_{max}, D_{max}))$	------	n^2mC_{max}
MTMP3	$nm(\min(C_{max}, nmD_{max}) + n\min(C_{max}, D_{max}))$	$n^2m^2D_{max}$	n^2mC_{max}
MTMP4	$n^3m\min(C_{max}, D_{max})$	n^3mD_{max}	$n^3mC_{\max}$
MTMP5	$n^2m(\log n)\min(C_{max}, D_{max})$	$n^2m(\log n)D_{max}$	$n^2m(\log n)C_{max}$
FFMTMP	$nm(C_{max} + D_{max} + n\min(C_{max}, D_{max}))$	-----	-----
FMTMPR	$nm(\min(C_{max}, nmD_{max}) + D_{max} + n\min(C_{max}, D_{max}))$	$n^2m^2D_{max}$	-----
EKMTMP	$n((n\log n + m)\min(C_{max}, nmD_{max}) + nm\min(C_{max}, D_{max}))$	$n^2m(n\log n + m)D_{max}$	n^2mC_{max}

The Table 2 complexity expressions contain four or less network parameters and the only relationship among these parameters that we are free to assume is $n\text{-}1 \le m \le n(n\text{-}1)$, i.e., there are at least enough links for every node but s to have an incoming link and there are no more links than necessary to link each node to every other node in the network (where node u is linked to node v if and only if $(u, v) \in E$). Prior to making algorithm complexity comparisons based upon Table 2 expressions, we shall define terms that we shall use in the comparisons.

If $\forall\, k > 0$, $\Psi(n, m, C_{max}, D_{max}) \neq k\Omega(n, m, C_{max}, D_{max})$ and $\exists\ k_\Omega > 0$ such that $\forall$ possible set of n, m, C_{max} and D_{max}, $\Psi(n, m, C_{max}, D_{max}) < k_\Omega\Omega(n, m, C_{max}, D_{max})$, then $\Psi(n, m, C_{max}, D_{max})$ is "over bounded" by $\Omega(n, m, C_{max}, D_{max})$. Algorithm A is superior to (has a superior complexity than) Algorithm B and Algorithm B is inferior to (has an inferior complexity than) Algorithm A if Algorithm A has complexity $O(\Psi(n, m, C_{max}, D_{max}))$ and $\forall\ \Omega(\cdot,\cdot,\cdot,\cdot)$ such that Algorithm B has complexity $O(\Omega(n, m, C_{max}, D_{max}))$, $\Psi(n, m, C_{max}, D_{max})$ is over bounded by $\Omega(n, m, C_{max}, D_{max})$.

[34] MTMP4's routine, see Ahuja and Orlin (Ahuja and Orlin 1991) and MTMP5's routine, see Sleator and Tarjan (Sleator and Tarjan 1993) each use "distance" labels and a series of "advances" and "retreats" to find augmenting generalized paths from s to t. In addition MTMP5's routine uses a "dynamic tree" data structure to reduce the average number of operations required to find an augmenting generalized path. Neither of these routines, unlike the ones used in MTMP2 (and FFMTMP) and MTMP3 (and EKMTMP), has the property that a "best" complexity can be found by multiplying an upper bound on the number of flow augmenting generalized paths by a complexity per augmentation. Each augmentation carries computational overhead where the sum of these overheads can be quantified over the set of augmentations to provide a lower complexity value than can be obtained by multiplying a maximum complexity of any augmentation by the number of augmentations.

Let $\Psi(n, m, C_{max}, D_{max}) = \sum_{j=1}^{q} \psi_i(n, m, C_{max}, D_{max})$ and let the limit as $n \rightarrow \infty$ of the fraction of the range of m, where m's range is $[n - 1, n(n - 1)]$, for which for some j, $1 \leq j \leq q$, $O(\Psi(n, m, C_{max}, D_{max})) = O(\psi_j(n, m, C_{max}, D_{max}))$ be 1 (i.e., $\forall i$: $1 \leq i \leq q$ and $i \neq j$ and $\forall k > 0$, the fraction of m's range for which $k\psi_i(n, m, C_{max}, D_{max}) \leq \psi_j(n, m, C_{max}, D_{max})$ has a limit of 1 as $n \rightarrow \infty$). Further let there be no set of linearly independent functions $\psi_{j,i}(n, m, C_{max}, D_{max})$, $1 \leq i \leq p$ and $p \geq 2$, such that $\psi_j(n, m, C_{max}, D_{max}) = \sum_{j=1}^{q} \psi_i(n, m, C_{max}, D_{max})$ and for some l, $1 \leq l \leq p$, the limit as $n \rightarrow \infty$ of the fraction of m's range for which $O(\psi_j(n, m, C_{max}, D_{max})) = O(\psi_{j,l}(n, m, C_{max}, D_{max}))$ is 1. Then $O(\psi_j(n, m, C_{max}, D_{max}))$ is an asymptotic value of $O(\Psi(n, m, C_{max}, D_{max}))$. Algorithm A is asymptotically superior to Algorithm B and Algorithm B is asymptotically inferior to Algorithm A if $O(\hat{\Psi}(n, m, C_{max}, D_{max}))$ is an asymptotic value of an Algorithm A complexity expression and $\forall \hat{\Omega}(\cdot,\cdot,\cdot,\cdot)$ for which $O(\hat{\Omega}(n, m, C_{max}, D_{max}))$ is an asymptotic value of an Algorithm B complexity expression, $\hat{\Psi}(n, m, C_{max}, D_{max})$ is over bounded by $\hat{\Omega}(n, m, C_{max}, D_{max})$.

We shall in our comparisons consider cases in which algorithm complexities are restricted to being expressed in limited fashions (per 2's columns 3 and 4). We shall use the terms superior, inferior, asymptotically superior and asymptotically inferior in these limited cases in the same manner as for the "full" complexities, i.e., the case for which the terms were defined above.

Only MTMP1 and EKMTMP in Table 2 have asymptotic complexities that are different from their complexities. MTMP1's asymptotic complexities are respectively $O(n^2 m \min(C_{max}, D_{max}))$, $O(n^2 m D_{max})$ and $O(n^2 m C_{max})$ when complexities are unlimited, i.e., full, limited to being expressed independent of C_{max} and limited to being expressed independent of D_{max}. This can be seen by letting $\varepsilon(n)$ be the percentage of the range of m that lies between n -1 and $n^2/\log n$. Then $\lim_{n\rightarrow\infty} \varepsilon(n) = 0$, i.e., the percentage of the range of m in which $m < n^2/\log n$ (corresponding to the percentage of the range for which MTMP1's first complexity expression term is greater than its second term, e.g., $(n^4/\log n) C_{max} > n^2 m C_{max}$ in the case where the complexity is expressed independent of D_{max}), has a limit of zero as $n \rightarrow \infty$[35]. EKMTMP's asymptotic complexities are $O(nm(\min(C_{max}, nmD_{max}) + n\min(C_{max}, D_{max}))$ and $O(n^2 m^2 D_{max})$ when complexities are unlimited and limited to being expressed independent of C_{max} respectively[36].

Based upon the "full" expressions of the algorithm complexities there is no Table 2 generally superior algorithm. However MTMP1 (the only one of the routines that doesn't use a flow augmenting generalized path approach) is asymptotically superior to all the other algorithms. When complexity expressions are limited to be independent of C_{max}, MTMP1 is superior to all but MTMP5 and is asymptotically superior to all the other algorithms including MTMP5. Further when complexities are expressed independent of D_{max}, while MTMP1 is inferior to some of the other algorithms, the "best" of them aren't asymptotically superior to MTMP1.

---

[35] $\varepsilon(n) = \frac{(n^2/\log n)-(n-1)}{n(n-1)-(n-1)} = \frac{(n^2/\log n)-(n-1)}{(n-1)^2}$. Hence $\lim_{n\rightarrow\infty} \varepsilon(n) = \lim_{n\rightarrow\infty} \frac{(n^2/\log n)-n}{n^2} = \lim_{n\rightarrow\infty} \left(\frac{1}{\log n} - \frac{1}{n}\right) = 0$.

[36] Here we set $\hat{\varepsilon}(n) = \frac{n\log n-(n-1)}{n(n-1)-(n-1)} = \frac{n\log n-(n-1)}{(n-1)^2}$. Hence $\lim_{n\rightarrow\infty} \hat{\varepsilon}(n) = \lim_{n\rightarrow\infty} \frac{(n\log n)-n}{n^2} = \lim_{n\rightarrow\infty} \left(\frac{\log n}{n} - \frac{1}{n}\right) = 0$.

APPENDIX B. MTMP ALGORITHM CHOICES

We shall consider in this appendix factors related to our choices of algorithms for Min_Path and Max_Flow. We chose to use Dijkstra's algorithm as the major part of Min_Path. This necessitated assigning augmented residual network, N_R, link delays that were more involved than might otherwise have been required. We could have simply set the delays of non-zero capacity N_R links to plus and minus the original network, N, link delays, i.e., links of the form (u'', v') and (v', u'') with non-zero capacity would be assigned link delays $D_R(u'', v') = D(u, v)$ and $D_R(v', u'') = -D(u, v)$ respectively. However then a Min_Path built around Dijkstra's algorithm that requires non-negative delays wouldn't have been possible. Instead we could have used the Bellman-Ford algorithm for Min_Path provided that we could show that with these link delay assignments N_R doesn't have a negative delay loop, i.e., a loop in which the sum of the loop's link delays is < 0. We shall demonstrate the non-existence of negative delay loops in the proof of the following lemma. Here the simple link delay assignment refers to assigning $D(u, v)$ to $D_R(u'', v')$ rather than $\hat{\delta}_R(u'') - \hat{\delta}_R(v') + D(u, v)$ and $-D(u, v)$ to $D_R(v', u'')$ rather than $\hat{\delta}_R(v') - \hat{\delta}_R(u'') - D(u, v)$ for nonzero capacity links of the form (u'', v') and (v', u'') respectively in N_R.

Lemma 4: No N_R network produced by MTMP has a negative delay loop when the simple link delay assignment is applied to N_R.

Proof: Since the first N_R produced by MTMP is generated from N, it has no links with negative delays and hence cannot have a negative delay loop (all (v', u'') type links in N_R have zero capacity and hence are assigned delays of ∞ rather than $-D(u, v)$, (see Figure 2 in Subsection 5.2 on page 17 with $\hat{D}_R(u, v) = D(u, v)$ and $\hat{D}_R(v, u) = D(v, u)$). Let *LOOP* represented by the sequence of links $(x_0,x_1), (x_1,x_2), \ldots, (x_{p-1}, x_p), (x_p, x_{p+1})$ where $x_{p+1} = x_0$ be a loop in N_R. If any link in *LOOP* has infinite delay, then LOOP has infinite delay and hence doesn't have a negative delay. Therefore assume all links in *LOOP* have a finite delay. Then every link in the loop has a non-zero capacity since all links with zero capacity in N_R have delays of ∞. Construct *GLOOP* in N_f corresponding to *LOOP* in N_R (recognizing per Figure 2 that each node x_i in N_R was generated by a node y_i in N_f and either has the form y_i' or y_i'') by creating a sequence of links in N_f (in the order of *LOOP*'s sequence of links) as follows:

include link $(y_i, y_{i+1}, 1)$	when x_i was generated from y_i and is of the form y_i'' and x_{i+1} was generated from y_{i+1} and is of the form y_{i+1}'
include link $(y_i, y_{i+1}, -1)$	when x_i was generated from y_i and is of the form y_i' and x_{i+1} was generated from y_{i+1} and is of the form y_{i+1}''
include no corresponding link	when x_i and x_{i+1} were generated from the same node in N_f

After this construction the resulting sequence is a generalized loop in N_f and the delay of *GLOOP* is (given the relation of delays in links of N_R and N_f, including that links of the form (y_i'', y_i') and (y_i', y_i'') in N_R have zero delay):

$$D(GLOOP) = \sum_{(y_i, y_{i+1}, d_{i+1}) \in GLOOP} \tilde{D}(y_i, y_{i+1}, d_{i+1}) = \sum_{i=1}^{p} D_R(x_i, x_{i+1}) = D(LOOP) \qquad (68)$$

Every link in *GLOOP* (given the relationship of link capacities in N_R and N_f) has non-zero capacity in the direction of *GLOOP* and hence *GLOOP* is a non-zero capacity generalized loop in N_f.

Since for some $T > 0$, f maximizes $D(\cdot, T)$, by Lemma 3 in Section 3 on page 10, $D(GLOOP) \geq 0$. Hence by equation (68), $D(LOOP) \geq 0$ and therefore N_R has no negative delay loops. ■

The Bellman-Ford algorithm whose complexity is O(nm) is inferior to Dijkstra's algorithm whose complexity of O(m + $n$log$n$). However Bellman-Ford's use wouldn't increase the complexity of MTMP since Decompose_to_Path_Flows also has complexity O(nm). Our choice of a Min_Path built around Dijkstra's algorithm however causes Decompose_to_Path_Flows to be a dominant factor in MTMP's complexity and hence assures that any reduction of Decompose_to_Path_Flows' complexity will improve MTMP's complexity (see Appendix D).

Our choice of Max_Flow algorithms was targeted at two separate cases. The first case is one in which link delays are all non-negative integers (and link capacities are only constrained to be positive) and the second case is one in which link capacities are all positive integers (and link delays are only constrained to be non-negative). The first case lead to MTMP1 in which we chose the maximum flow algorithm of Cheriyan Maheshwari (Cheriyan and Maheshwari 1989) that, among those we are aware of, has the "best" complexity when complexities are limited to being expressed only in terms of n and m. This maximum flow algorithm as noted in Table 1 in Subsection 5.3 on page 21 has complexity O(n^3/log$n$) and is neither inferior nor superior to Decompose_to_Path_Flows whose complexity is O(nm). Hence both of these algorithms impact MTMP1's complexity that is O($n^2(n^2$/log$n$ + $m)D_{max}$) even when link capacities aren't integers. As noted in Subsection A.6 on page A-13, this MTPMP1 complexity reduces to an asymptotic complexity of O(n^2mD_{max}) determined by Decompose_to_Path_Flows' complexity(which, per the definition in the last paragraph of this appendix, results from MTMP1's maximum flow algorithm being superior in the limit to Decompose_to_Path_Flows').

The second case's integer link capacity requirement leads to all flow augmenting generalized paths (used in flow maximizations) being assigned integer flow rates. This, in turn, causes the number of flow augmenting generalized paths used to be bounded above by the maximum network flow that is less than nC_{max}. The most straightforward flow augmenting generalized path maximum flow algorithm is such that the complexity of the totality of its executions by MTMP is O(nmC_{max}). That complexity is equal to the complexity of the totality of executions of other MTMP operations (i.e., those in Table 1 whose single execution complexities are O(m)) that are all dominated by the totality of Decompose_to_Path_Flows' executions. Hence the straight forward flow augmenting generalized path maximum flow algorithm was chosen for MTMP2 causing MTMP2's complexity, when only its capacities have to be integers, to be determined by the totality of Decompose_to_Path_Flows' executions which is O(n^2mC_{max}).

The "best" maximum flow algorithm for MTMP1 was selected based in part on a comparison related to asymptotic complexity discussed in Subsection A.6 on page A-13. Its O(n^3/log$n$) complexity makes it superior to a number of the strictly comparable algorithms (i.e., other algorithms whose complexity can be expressed only in terms of n and m), e.g., a "preflow push to front" algorithm with complexity O(n^3), see Goldberg (Goldberg, 1985). However there exists a set of strictly comparable algorithms that this best routine is neither superior to nor asymptotically superior to. We define another ratio based limit comparison to distinguish among these. Particularly we say that if the limit as $n \rightarrow \infty$ of the fraction of the range of m for which algorithm A has lower complexity than algorithm B is 1, then algorithm A is superior in the limit to algorithm B. MTMP1's maximum flow routine is superior in the limit to all the strictly comparable maximum flow routines that we are aware of. For example, one strictly comparable algorithm's complexity is O($n^2\sqrt{m}$) but isn't O(n^3/ log$n$), see Cheriyan and Maheshwari (Cheriyan and Maheshwari 1989). The chosen algorithm therefore has a lower complexity when n^2/log$^2n < m$. The fraction

of the range of m for which $n^2/\log^2 n > m$ is $\frac{n^2/\log^2 n-(n-1)}{(n-1)^2}$ whose limit as $n \to \infty$ is zero. So the limit of the fraction of the range for which $n^2/\log^2 n < m$ is 1 and therefore the chosen routine is superior in the limit[37].

---

[37] A routine for which the fractional limit requires a more involved computation is one whose complexity is $O(nm\log\frac{n^2}{m})$ but not $O(n^3/\log n)$, see Goldberg and Tarjan (Goldberg, and Tarjan 1986). Here in comparing its complexity to the chosen algorithm's complexity, the comparison is between $m\log\frac{n^2}{m}$ and $\frac{n^2}{\log n}$. $m\log\frac{n^2}{m}$ has a maximum at $m = n^2/e$, equals $n^2/2$ at $m = n^2/2$, equals $(n\text{-}1)\log\frac{n^2}{n-1}$ at the minimum m of n - 1, equals $n(n\text{-}1)\log\frac{n}{n-1}$ at the maximum m of $n(n$ - 1), is 0 at $m = n^2$, has positive slope in $[(n\text{-}1), n^2/e)$ and has negative slope in $(n^2/e, n^2]$. Thus for n sufficiently large there is an initial interval from n-1 to a value less that $n^2/2$ for which $\frac{n^2}{\log n} > m\log\frac{n^2}{m}$, a second interval containing $n^2/2$ for which $\frac{n^2}{\log n} < m\log\frac{n^2}{m}$ and possibly a third interval from a value greater than $n^2/2$ to $n(n\text{-}1)$ for which $\frac{n^2}{\log n} > m\log\frac{n^2}{m}$ since such a third interval exists within $(n^2/2, n^2)$.

Since for $m < n^2/2$, $m\log\frac{n^2}{m} > m$, the interval from n-1 to $n^2/2$ for which $m\log\frac{n^2}{m} < \frac{n^2}{\log n}$ is contained in the interval in which $m < \frac{n^2}{\log n}$. Therefore, since $\lim_{n\to\infty} \frac{n^2/\log n-(n-1)}{(n-1)^2} = 0$, the ratio of first of the three intervals length to the range of m has a limit of 0 as $n \to \infty$.

If we let $n(\varepsilon)$ be the value of n at which $\frac{n^2}{\log n} = m\log\frac{n^2}{m}$ when $m = (1 - \varepsilon)n(\varepsilon)^2$, then $n(\varepsilon) = 2^{\left(\frac{1}{(1-\varepsilon)\log((1-\varepsilon)^{-1})}\right)}$, $\lim_{\varepsilon\to 0} n(\varepsilon) = \infty$, and for $n(\varepsilon)^2/2 < m < (1 - \varepsilon)n(\varepsilon)^2$, $m\log\frac{n(\varepsilon)^2}{m} > \frac{n(\varepsilon)^2}{\log n}$. The third interval from $(1- \varepsilon(n))n^2$ to $n(n\text{-}1)$, where $\varepsilon(n)$ is the inverse of $n(\varepsilon)$ for $n > 4$ (corresponding to $0 < \varepsilon < 1/2$), is contained in the interval from $(1- \varepsilon(n))n^2$ to n^2. Hence the third interval has a length $< n^2\text{-}(1- \varepsilon(n))n^2$. Note that $n(\varepsilon)$ has an inverse for $n > 4$ since $n(\varepsilon)$ has a negative derivative in the range $0 \le \varepsilon < 1/2$. Also note that $\lim_{n\to\infty} \varepsilon(n) = 0$. Therefore the limit as $n \to \infty$ of the ratio of the third interval to the range of m is $\le \lim_{n\to\infty} (n^2\text{-}(1- \varepsilon(n))n^2)/n(n\text{-}1) = \lim_{n\to\infty} \varepsilon(n) = 0$.

Since the limits as $n \to \infty$ of the ratios of the first and third intervals to the range of m are both zero, the limit of the ratio of the interval for which $n^3/\log n < nm\log\frac{n^2}{m}$ is 1. Hence MTMP1's maximum flow algorithm is superior in the limit to the algorithm whose complexity is $O(nm\log\frac{n^2}{m})$.

APPENDIX C. INTEGER MESSAGE SEGMENTS REQUIREMENT

C.1 IMPACT ON QUICKEST MULTIPATHS

The path table generation discussed in the body of this report allowed messages to be arbitrarily divided among paths of a multipath (i.e., a path's segment did not have to be an integer number of message units). We shall discuss in this appendix the impact of requiring integer length path segment assignments. We shall refer to the situation allowing non-integer segments as the unrestricted case and refer to the situation requiring only integer segments as the restricted case.

The definition of multipath *MP*'s, end-to-end delay for a message length σ can be generalized from equation (8), on page 3 in Section 1, to be:

$$T_{MP,\,\sigma} = \min_{\phi \in \Psi_\sigma} \left(\max_{P_i \in P:\sigma_i \neq 0} (\sigma_i / R(P_i) + D(P_i)) \right)$$

where $\varphi = (\sigma_1, \sigma_2, \ldots, \sigma_p)$, p is the number of paths in P (where $MP = (P, R)$), $D(P_i)$ and $R(P_i)$ are respectively path P_i's delay and MP flow rate and $\Psi_\sigma = \{\varphi$ satisfying a set of conditions: $\sum_{i=1}^{p} \sigma_i = \sigma \}$. This definition provides a multipath's unrestricted case end-to-end delay when the set of conditions defining Ψ_σ is $\forall\ i,\ i = 1, 2, \ldots, p,\ \sigma_i \geq 0$ and its restricted case end-to-end delay when the set of conditions defining Ψ_σ is $\forall\ i,\ i = 1, 2, \ldots, p,\ \sigma_i$ is a non-negative integer. The multipath unrestricted case end-to-end delay is defined by a unique segment assignment φ in which the end-to-end delay of all paths with non-zero segments is equal to the multipath end-to-end delay (i.e., $\forall\ P_i \in MP:\ \sigma_i > 0,\ \sigma_i/R(P_i) + D(P_i) = T_{MP,\,\sigma}$). However such a segment assignment isn't always possible in the restricted case.

It is obvious that the unrestricted case minimum end-to-end delay for a message is a lower bound for the restricted case minimum end-to-end delay. The multipath $MP_1 = ((P_1),(6)) = (([s, u, t]), (6))$ in Figure 3 on page C-2 is an unrestricted case quickest multipath for any message length σ in the range $0 < \sigma \leq 6$ where each such σ corresponds to a minimum end-to-end delay T in the range $3 < T \leq 4$. $MP_2 = ((P_1, P_2, P_3), (6, 6,6)) = (([s, u, t], [s, v, w, y, t], [s, v, x, y, t]), (6, 6, 6))$ is an unrestricted case quickest multipath for $\sigma \geq 6$ corresponding to $T \geq 4$. MP_1 is a restricted case quickest multipath for integer message lengths of 1 to 6. However MP_2 is not a restricted case quickest multipath for all integer $\sigma \geq 6$. It is a restricted case quickest multipath for message lengths σ = 6, 9, 12, 15, 18, 21, 24, 27 ... corresponding to $T = 4, 4\frac{1}{6}, 4\frac{1}{3}, 4\frac{1}{2}, 4\frac{2}{3}, 4\frac{5}{6}, 5, 5\frac{1}{6}$, ... having the same restricted case as unrestricted case end-to-end delays (since unrestricted case message segment assignments are all integers for these message lengths). For $\sigma = 10$, MP_2's unrestricted case end-to-end delay is $4\frac{2}{9}$ but its restricted case end-to-end delay is $4\frac{1}{3}$ (with e.g., 7, 2 and 1 message units being assigned respectively to P_1, P_2 and P_3). MP_2 is not a restricted case quickest multipath for $\sigma = 10$, for consider the multipath $MP_3 = ((P_1, P_2, P_3), (6, 8, 4))$ which is also an unrestricted case quickest multipath for $\sigma \geq 6$. Its restricted case end-to-end delay for $\sigma = 10$ is $4\frac{1}{4}$ (with 7, 2 and 1 message units being assigned respectively to P_1, P_2 and P_3). MP_2 is also a restricted case quickest multipath for σ = 8, 11, 14, ... However its restricted case end-to-end delay for these message lengths is greater than its unrestricted case end-to-end delay.

MP_3 is a restricted case quickest multipath for σ = 6, 15, 24, 33, ... corresponding to $T = 4, 4\frac{1}{2}, 5, 5\frac{1}{2}$, ... having the same restricted case as unrestricted case end-to-end delays. However it is not a restricted case quickest multipath for $\sigma = 12$. Its restricted case minimum end-to-end delay for $\sigma = 12$ is $4\frac{3}{8}$ (with 8, 3, and 1 message units being assigned respectively to P_1, P_2 and P_3) while MP_2's is $4\frac{1}{3}$ (with 8, 2, and 2 message units

being assigned respectively to P_1, P_2 and P_3). MP_3 is also a restricted case quickest multipath for message lengths $\sigma = 7$ and 10 where in each case its restricted case end-to-end delay is greater than its unrestricted case end-to-end delay.

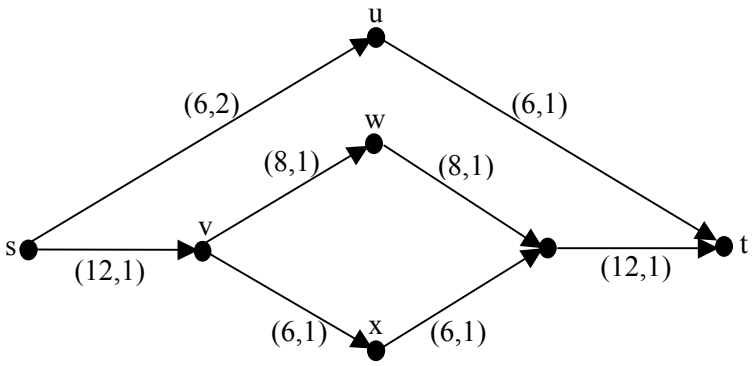

Fig. 3. First network for illustrating integer message segment requirement impacts.

A restricted case quickest multipath for a message length isn't necessarily an unrestricted case quickest multipath for that message length. Consider $MP_a = ([s, u, v, t], (5))$ and $MP_b = ([s, u, v, t], [s, v, t], [s, u, t]), (4, 1, 1))$ in Figure 4 below. MP_a is the restricted case quickest multipath for $\sigma = 6$ with an end-to-end delay of $4\frac{1}{5}$. MP_b's restricted case end-to-end delay for $\sigma = 6$ is $4\frac{1}{4}$. However MP_a is not an unrestricted case quickest multipath for $\sigma = 6$. MP_b whose unrestricted case end-to-end delay for $\sigma = 6$ is $4\frac{1}{6}$ is the unrestricted case quickest multipath for $\sigma > 5$.

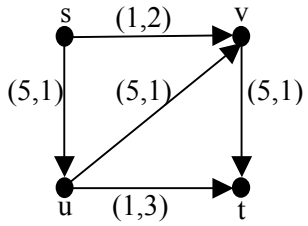

Fig. 4. Second network for illustrating integer message segment requirement impacts.

An unrestricted case quickest multipath path table produced by an MTMP algorithm contains a finite number of multipaths and each such multipath is a quickest multipath for a single continuous interval of message lengths. Table 3 on page C-3, in which $X = 5,6, 7, \ldots,$ lists a complete set of the Figure 3 network's restricted case quickest multipaths for integer message lengths[38]. It also includes the restricted case quickest multipath

[38] The table was developed by noting that the maximum number of message units that can be "communicated" over P_1 in time T is $X_1 = 6(T-3)$ and over the combination of P_2 and P_3 is $X_2 + X_3 = 12(T-4)$. Hence for an integer message length $\sigma > 6$, $3\frac{2}{3} + \frac{\sigma}{18}$ is a lower end-to-end delay bound where $X_1 = \frac{\sigma}{3} + 4$ and $X_2 + X_3 = \frac{2\sigma}{3} - 4$. X_1 is an integer and $X_2 + X_3$ is an even integer for $\sigma = 9, 12, 15, \ldots$, so $MP_2 = ((P_1,P_2, P_3), (6,6,6))$ is a restricted case quickest multipath for these message lengths. For other integer $\sigma > 6$, X_1 or X_2+X_3 has to be increased with the other reduced accordingly so X_1 and $X_2 + X_3$ are integers to obtain a quickest restricted case multipath. For $\sigma = 8, 11, 14, \ldots$, increasing either to the next higher integer adds $\frac{1}{18}$ to the end-to-end delay so MP_2 is a restricted case quickest multipath for these message lengths also. For $\sigma = 7, 10, 13, \ldots$, increasing X_1 to the next higher integer adds $\frac{1}{9}$ while increasing $X_2 + X_3$ to the next higher integer adds $\frac{1}{36}$ to the end-to-end delay, so $X_2 + X_3$ is increased for a restricted case quickest multipath. The table's values for X_2 and X_3 and flow rates for P_2 and P_3 given the increased $X_2 + X_3$ lead to equal end-to-end delays for P_2 and P_3 thereby minimizing the end-to-end delay for the message lengths $\sigma = 10, 13, 16, 19, \ldots$ (with $\sigma = 7$ being a special case).

segment assignments and minimum end-to-end delays for the associated integer message lengths. We note the following aspects of the table:

1. it has an infinite number of multipaths
2. it has multipath flow rates that are non-integer for message lengths σ = 13, 16, 19, …, despite the integer capacities of the network links
3. multipaths are not limited to be quickest multipaths for "uninterrupted" ranges of integer message lengths

Table 3. Quickest figure 3 network multipaths

message length	quickest multipath	segment assignments	end-to-end delay
1	$((P_1), (6))$	1	$3\frac{1}{6}$
2	$((P_1), (6))$	2	$3\frac{1}{3}$
3	$((P_1), (6))$	3	$3\frac{1}{2}$
4	$((P_1), (6))$	4	$3\frac{2}{3}$
5	$((P_1), (6))$	5	$3\frac{5}{6}$
6	$((P_1), (6))$	6	4
7	$((P_1, P_2), (6, 8))$	6, 1	$4\frac{1}{8}$
8	$((P_1, P_2, P_3), (6, 6, 6))$	6, 1, 1	$4\frac{1}{6}$
9	$((P_1, P_2, P_3), (6, 6, 6))$	7, 1, 1	$4\frac{1}{6}$
10	$((P_1, P_2, P_3), (6, 8, 4))$	7, 2, 1	$4\frac{1}{4}$
11	$((P_1, P_2, P_3), (6, 6, 6))$	8, 2, 1	$4\frac{1}{3}$
12	$((P_1, P_2, P_3), (6, 6, 6))$	8, 2, 2	$4\frac{1}{3}$
13	$((P_1, P_2, P_3), (6, \frac{36}{5}, \frac{24}{5}))$	8, 3, 2	$4\frac{5}{12}$
. . .	. . .	. . .	. . .
$3X$ - 2	$((P_1, P_2, P_3), (6, \frac{12(X-2)}{2X-5}, \frac{12(X-3)}{2X-5}))$	$X+3, X-2, X-3$	$\frac{2X+43}{12}$
$3X-1$	$((P_1, P_2, P_3), (6, 6, 6))$	$X+4, X-2, X-3$	$\frac{X+22}{6}$
$3X$	$((P_1, P_2, P_3), (6, 6, 6))$	$X+4, X-2, X-2$	$\frac{X+22}{6}$
. . .	. . .	. . .	. . .

All of the aspects noted are contrasts to a MTMP generated unrestricted case path table (see Table 4 on page C-4). The infinite number of multipaths results from different quickest multipaths for message lengths of the form $3X$ - 2, for[39] $X \geq 5$.

[39] A quickest restricted case multipath for σ = 13, 16, 19, ... requires that P_2 and P_3 together communicate $(2(\sigma+2)/3)$ - 5 message units and that P_2 and P_3's flow rates and message assignments lead to these paths having equal end-to-end delays. This requires that P_2's flow rate be $12X_2/(2X-5)$, where $X = (\sigma+2)/3$ and, in order for the sum of P_2 and P_3's flow rates to be 12 without either exceeding its path capacity, $(2X-5)/2 \leq X_2 \leq 2(2X-5)/3$. Hence $X_2 < 2X-5$. If σ is such that $2X$ - 5 is a prime ≥ 5, then, since $X_2 < 2X$ - 5, $2X$ - 5 can't be a factor of $12X_2$ (for otherwise $2X$ - 5 would be a product of proper factors of $12X_2$). Therefore the flow rate cannot be reduced to a fraction with a smaller denominator than $2X$ - 5. Hence if X is an integer ≥ 5 and $2X$ - 5 is a prime, then no restricted case quickest multipath for a message length other than $\sigma = 3X$ -2 has a flow rate $12X_2/(2X-5)$. The range of $2X$ - 5 for $X \geq 5$ includes all prime numbers ≥ 5. Therefore there must be an infinite number of multipaths in the table.

We shall conclude this subsection by stating a sufficient condition for a MTMP generated unrestricted case quickest multipath to be a restricted case multipath in the situation where all of the network link delays are non-negative integers and all network link capacities are positive integers. The flow rate and delay of any path of a multipath generated by the MTMP algorithms are positive integers in this situation. Hence for an integer time greater than its path delay, such a path will transmit and have received at its terminal node (i.e., communicate) an integer number of message units. Further the end point times of the end-to-end delay interval over which the multipath is an unrestricted case quickest multipath correspond to generalized path delays and therefore are integers. Hence an integer length message whose end-to-end delay over the multipath is equal to an end-point time for the multipath can be transmitted over the multipath in integer length segments.

Suppose $MP = ((P_1, P_2, ..., P_r), (R(P_1), R(P_2), ..., R(P_r))$ is a MTMP generated quickest unrestricted case multipath for σ_{e1} in a network whose link delays are all non-negative integers and link capacities are all positive integers. Further suppose that MP's end-to-end delay for σ_{e1} is T_{e1}, where T_{e1} is the lower end-point time for MP, $R(P_i) = a_iM$ and a_i is a positive integer for $i = 1, 2, ..., r$, M is a positive integer and is the greatest common factor of the $R(P_i)$ and $a = \sum_{i=1}^{r} a_i$. Then MP will be able to communicate a message of length $\sigma_{e1} + pa$ in integer segments with end-to-end delay of $T_{e1} + p/M$, where p is any positive integer. Thus if $\sigma_{e1} + pa$ is in the range of message lengths for which MP is an unrestricted case quickest multipath, then MP is a restricted case quickest multipath for $\sigma_{e1} + pa$ (for MP_2 in Figure 3 on page C-2, $M = 6$ and $a = 3$ while for MP_3, $M = 2$ and $a = 9$).

C.2 COMPLEXITY

The MTMP algorithms (applicable to the unrestricted case) effectively produce a table like Table 4 on this page. The more complicated nature of Table 3 on page C-3 (e.g., where multipaths are quickest multipaths for more than a single message length interval) suggests that problems requiring restricted case solutions may have greater complexity than those allowing unrestricted case solutions.

Table 4. MTMP produced table

message length	quickest time	quickest multipath
0	T_0	Nill
σ_1	T_1	MP_1
σ_2	T_2	MP_2
. . .	. . .	. . .
σ_{max}	T_{max}	MP_{max}
no entry	no entry	$MP_{max + 1}$

We define the path table multipath problem, PTMPP, (solved by the MTMP algorithm) as follows: Given a network N and nodes s and t in N, develop a table with message length, time and multipath (from s to t) columns where:

1. the first row's message length is zero and time is the minimum path delay from s to t
2. in any row, other than the first and last row (if there is a last row), the multipath is an unrestricted case quickest multipath for this row's message length and all message lengths between the previous row's message length and this row's message length
3. if the table is finite, the last row's multipath is an unrestricted case quickest multipath for any message length greater than the next to last row's message length
4. the time in each row other than the first and last row is the unrestricted case quickest multipath end-to-end delay for the row's message length

We shall refer to a corresponding problem requiring a restricted case solution as the restricted path table multipath problem, RPTMPP, and define it as follows: Given a network N and nodes s and t in N, develop a table with message length, time and multipath (from s to t) columns where:

1. the first row's message length is zero and time is the minimum path delay from s to t
2. in any row, other than the first and last row (if there is a last row) each row's message length is an integer, its multipath is a quickest restricted case multipath for the row's message length and for all integer message lengths that are less than or equal to this row's and greater than the previous row's message length
3. if the table is finite, the last row's multipath is a quickest restricted case multipath for any integer message length greater than the next to last row's message length
4. the time in each row other than the first or last row is the quickest restricted case multipath end-to-end delay for the row's message length.

Corresponding to the MTMPP (defined in Section 1 on page 3) we define the restricted minimum time multipath problem, RMTMPP to be: Given a network N and nodes s and t in N, and a positive integer message length, find a restricted case quickest multipath from s to t for the message length. The "decision version" of MTMPP, which we shall refer to as the bounded time multipath problem, BTMPP, is the problem: Given a network N, two nodes s and t in N, a positive message length, σ, and a positive time T, determine if there is a multipath in N from s to t which has an end-to-end delay for σ that is $\leq T$ and, if there is, determine one such multipath. Its corresponding restricted case problem, the restricted case bounded time multipath problem, RBTMPP, is the problem: Given a network N, two nodes s and t in N, a positive integer message length, σ, and a positive time T, determine if there is a restricted case multipath in N from s to t which has an end-to-end delay for σ that is $\leq T$ and, if there is, then determine one such multipath.

We shall in the rest of this subsection limit consideration to networks having integer link capacities. Any PTMPP meeting this limitation will admit to a solution in which the number of table rows is $\leq nC_{max}$ (see Subsection 5.3, page 21). If in addition we restrict C_{max} so that $C_{max} \leq P(n,m)$, where $P(n,m)$ denotes a polynomial in n and m, then PTMPP can be solved in polynomial time (e.g., via the MTMP algorithm[40]). For any positive message length σ, the position of σ relative to the PTMPP table's message lengths can be found via a binary search type algorithm (a "squeeze algorithm"[41]) in $\log(nC_{max})$ time. Therefore (in networks with the above restrictions) an MTMPP can be solved via its corresponding PTMPP in polynomial time and a BTMPP can be solved via its corresponding MTMPP in polynomial time.

If a RPTMPP can be solved in polynomial time, then the table it produces must have a finite number of rows. Hence, through the reasoning corresponding to the unrestricted case, the corresponding RMTMPP and RBTMPP can be solved in polynomial time. Conversely if an RBTMPP cannot be solved in polynomial time then its corresponding RMTMPP and RPTMPP can't be solved in polynomial time. Since Kagaris et. al. proved that the RBTMPP (for integer link capacities) is NP-complete (Kagaris et. al. 1999), it can't be solved

[40] We in general follow Ahuja et. al. (Ahuja et. al. 1993) in considering a network oriented problem to be a polynomial time problem if it can be solved with an algorithm whose run time is bounded by a polynomial in n, m, $\log C_{max}$ and $\log D_{max}$ and a pseudopolynomial time problem to be one which can be solved with an algorithm whose run time is bounded by a polynomial in n, m, C_{max} and D_{max}. The more general notion of a polynomial time problem, as used for example in a later footnote, is a problem that can be solved by an algorithm whose complexity is $O(r^i)$ for some integer i, where r is the number of problem inputs.

[41] If $\sigma \geq \sigma_{max}$ then a quickest multipath can be found immediately. Otherwise set $I_L = 1$ and $I_H = i_{max}$ and successively do the following: set $I = \lfloor (1/2)(I_H + I_L) \rfloor$ and then, if $\sigma_I \geq \sigma$, set $I_H = I$ or otherwise set $I_L = I$. Halt when $\sigma_{I_H} = \sigma$ or $I_H = I_L + 1$. The quickest multipath is then MP_{I_H}.

in polynomial time (assuming P ≠ NP). Hence the RMTMPP and therefore the RPTMPP can't be solved in polynomial time[42].

The proof in Kagaris et. al.(Kagaris et. al.1999) reduces a known NP-complete problem[43] to an RBTMPP in which all link capacities are one and all link delays are $1/K$ where K is a positive integer ≥ 5. Thus the NP-complete result in the Kagaris reference doesn't directly apply to the subclass of RBTMPP problems in which the delays are additionally constrained to be integers[44] (though it might be considered suggestive that this subclass of problems is NP complete). However the result of the problem of Figure 3 on page C-2, shown in Table 3 on page C-3, demonstrates that all RPTMPPs in the class in which all link delays and capacities are respectively non-negative and positive integers and $C_{max} \leq P(n, m)$ are not solvable in polynomial time, since the required table is infinite in length[45].

Xue defined a problem that is closely related to the RMTMPP (Xue 2003). The following definitions are used in our statement of this problem.

1. N = a network whose link delays and capacities are all positive integers
2. S = { $\overline{MP}$ in N: $\overline{MP}$ is a multipath from node s to node t in N with integer flow rates}
3. $T(P_i, \sigma)$ = path P_i's end-to-end delay for message length σ
4. $P_1^{MP}, P_2^{MP}, ..., P_{p(MP)}^{MP}$ are multipath MP's paths
5. $\Psi(MP, \sigma) = \{(\sigma_1, \sigma_2, ..., \sigma_{p(MP)}): \sum_{i=1}^{p(MP)} \sigma_i = \sigma$ and σ_i is a non-negative integer $\forall\ i,\ i = 1, 2, ..., p(MP)\}$
6. $\delta(MP, \sigma) = \min_{(\sigma_1, \sigma_2, ..., \sigma_{p(MP)}) \in \Psi(MP, \sigma)} \left(\max_{i=1,2,..., p(MP)} T(P_i^{MP}, \sigma_i) \right)$

---

[42] Here: 1. NP is the class of problems for which a proposed problem solution can be checked for correctness in polynomial time. 2. a problem A can be reduced to a problem B, if A can be solved by a "small" set of calls to an algorithm that solves B, where a "small" set could mean e.g., a set whose number of elements has a polynomial bound. 3. any problem that is in NP and to which any other problem in NP can be reduced is NP complete. 4. P is the class of problems that can be solved in polynomial time - the conjecture that NP ≠ P is as yet unproven but usually accepted.

[43] This known NP-complete problem is: Given a graph $G(V,E)$, two nodes in the graph, s and t, and two integers J and K, where $5 \leq K < |V|$, are there at least J mutually edge-disjoint paths from s to t with every one of these paths having K or less links.

[44] If RPTMPPs in networks of the proof could be reduced to RPTMPPs in networks with non-negative integer link delays and integer positive link capacities, then it would follow that the RPTMPP in such networks is NP-complete. It is therefore relevant to consider two types of PTMPP equivalence classes in which the network graph is the same for each problem in a class and the link delays and link capacities of any two problems in the class are related by a single positive constant $K > 0$. For the first type, $\forall\ e_i \in E,\ D_2(e_i) = \frac{D_1(e_i)}{K}$ and $C_2(e_i) = KC_1(e_i)$, while for the second type $\forall\ e_i \in E$, $D_2(e_i) = \frac{D_1(e_i)}{K}$ and $C_2(e_i) = C_1(e_i)$, where $D_1(e_i)$ and $C_1(e_i)$ and $D_2(e_i)$ and $C_2(e_i)$ represent respectively the link e_i's delay and capacity for problem 1 and problem 2. A problem 1 path table can be converted to a problem 2 path table for the first type by multiplying all times by $\frac{1}{K}$ and all multipath path flow rates by K (leading to the same message segment assignments). For the second type the problem 1 path table times and message lengths are multiplied by $\frac{1}{K}$ to yield the problem 2 path table (yielding message length assignments that are $\frac{1}{K}$ times the problem 1 assignments). These equivalence classes however don't provide the required reduction. No problem with positive integer capacities and non-negative integer delays is type 1 equivalent to a problem for a network used in the proof in Kagaris et. al.(Kagaris et. al.1999) (since obtaining fractional delays from the first problem requires link capacities be > 1). Further the type 2 equivalence doesn't carry over to RPTMPPs since a multiplication to obtain fractional delays might convert integer to non-integer segment assignments.

[45] The requirement that the RPTMPP produce the table is crucial here. It might be argued that we have a solution to the Figure 3 network quickest path problem that can be expressed in finite terms, i.e., for $\sigma \leq 6$ MP_1 is a quickest multipath, for $\sigma > 6$ and either $\frac{\sigma}{3} = \left\lfloor \frac{\sigma}{3} \right\rfloor$ or $\frac{\sigma+1}{3} = \left\lfloor \frac{\sigma+1}{3} \right\rfloor$ MP_2 is a quickest multipath, and for $\sigma > 6$ and $\frac{\sigma+2}{3} = \left\lfloor \frac{\sigma+2}{3} \right\rfloor$, $((P_1, P_2, P_3), (6, 12\left(\frac{\sigma-4}{2\sigma-11}\right),\ 12\left(\frac{\sigma-7}{2\sigma-11}\right)))$ is a quickest multipath.

7. *MP*'s integer end-to-end delay for message length σ is $\lceil \delta(MP, \sigma) \rceil$

The problem is: Given a network *N* with nodes *s* and *t*, and an integer message length σ, find a multipath $MP \in S$ with minimum integer end-to-end delay.

Since the network delays as well as the network capacities in Xue's problem are limited to be integer values, the multipaths determined by an MTTP algorithm for such a network are all restricted case multipaths at integer end-to-end delays. Therefore its PTMPP solution provides solutions to the Xue problem of for all message lengths. If the condition requiring integer flow rates is removed, i.e., now $S = \{MP \text{ in } N: MP \text{ is a multipath from node } s \text{ to node } t \text{ in } N\}$, then the PTMPP solution still provides solutions to the problem (with integer flow rates). It follows from the above discussion that Xue's problem can be solved in pseudopolynomial time, or if $C_{max} \leq P(n,m)$ in polynomial time, via the PTMPP[46].

We note that a solution of Xue's problem isn't necessarily a solution of an RMTMPP. This can be seen from the network of Figure 3 on page C-2 e.g., by comparing solutions for a message length of 13. The multipath $MP_1 = ((P_1, P_2, P_3), (6, 6, 6))$, with a delay of $4\frac{7}{18}$ is an MTMPP solution. This multipath's end-to-end delay in the restricted case is $4\frac{1}{2}$ so it is a solution Xue's problem with an integer end-to-end delay of 5. However the multipath $((P_1, P_2, P_3), (6, 7.2, 4.8))$ is an RMTMPP solution with an end-to-end delay of $4\frac{5}{12}$ (see Table 3 on page C-3) which is less than MP_1's restricted case end-to-end delay.

C.3 ADJUSTING THE UNRESTRICTED CASE SOLUTION

A unrestricted case quickest multipath $MP = (P,R) = ((P_1, P_2, ..., P_r), (R(P_1), R(P_2), ..., R(P_r))$ for an integer length message might be considered an approximation to a restricted case quickest multipath. Here the path segment length assignments (i.e., the segment assignments) can be adjusted to be non-negative integers so the multipath communicates the message with an end-to-end delay equal to its restricted case end-to-end delay for the message. The multipath's restricted case end-to-end delay for the message will be less than $\max_{P_i \in P}(1/R(P_i))$ greater than the end-to-end delay of a restricted case quickest multipath for the message[47].

---

[46] Xue provided an algorithm using a "squeeze" technique that solves the problem in polynomial time where the polynomial bound includes a $\log \sigma$ term (Xue 2003). Here he assumed that a set of general minimal cost flow problems, for integer times, are solved by a polynomial time linear programming routine. We can modify his algorithm so the resulting algorithm has a bounding polynomial in *n*, *m* and $\log D_{max}$, i.e. is independent of σ. We do this by letting its initial larger time be nD_{max} rather than the sum of σ and the shortest path delay, and having it recognize that if $\sigma > \sigma_{max}$, where σ_{max} is the maximum message length that can be communicated in nD_{max}, then a solution multipath is the same as a solution multipath for σ_{max}.

[47] Suppose $MP = (P, R)$ is an unrestricted case quickest multipath for the integer message length σ that communicates such a message in time T^* and has two or more non-integer assignments for σ. Further suppose that after the assignments are rounded up they sum to σ_1. Rounding up adds less than $\max_{P_i \in P}(1/R(P_i))$ to *MP*'s end-to-end delay and $\sigma_1 > \sigma$. Now suppose that MP's restricted case end-to-end delay for a message of length σ_1 is T_1 and $\hat{T}$ is the end-to-end delay of a restricted case quickest multipath for σ, then $\hat{T} \leq T_1 < T^* + \max_{P_i \in P}(1/R(P_i))$. Hence, since the end-to-end delay of a unrestricted case multipath for a message is a lower bound for the end-to-end delay of a restricted case multipath for the message, $T^* \leq \hat{T} < T^* + \max_{P_i \in P}(1/R(P_i))$. We note further that when *MP* is such that there is a unique maximum $1/R(P_i)$, then the restricted case quickest multipath end-to-end delay is bounded by $\max_{P_j \in P: P_j \neq P_i}(1/R(P_j))$. Therefore in this case $T^* \leq \hat{T} < T^* + \max_{P_j \in P: P_j \neq P_i}(1/R(P_j))$.

We discuss in this subsection an algorithm, Adjust, shown below on this page, that returns such adjusted segment assignments and the increase in the multipath's end-to-end delay resulting from the adjustment (though from the discussion of the network of Figure 3 on page C-2 it is clear that the resulting multipath isn't necessarily a restricted case quickest multipath).

Algorithm **Adjust**(*MP, L*)
1. **if** $\exists\ P_i \in P$ such that $L(P_i)$ isn't an integer **do**
2. Call Fractional_Adjust(*MP, L, τ, LIST, ΔT*);
3. Call Integer_Adjust(*MP, L, τ, LIST, ΔT*);
4. return (*L, ΔT*);

Adjust starts with a multipath, *MP* = (*P, R*), and a set of message assignments, *L*. If the message assignments aren't all integers, it calls two subroutines, Fractional_Adjust shown below on this page and Integer_Adjust shown on page C-9. The first of these subroutines adds fractional message units (up to one full message unit) to and subtracts fractional message units from unrestricted segment assignments to achieve all integer assignments. It does it in a manner that provides the minimum end-to-end delay that the multipath can achieve for the message when each multipath path's segment assignment is an integer that differs by no more than one from its unrestricted case segment assignment. The second subroutine adds and subtracts integer values to the segment assignments to achieve the best end-to-end delay that the multipath can achieve for the message for any all integer segment assignments. (Note that $\hat{\phi}$ in Integer_Adjust is the empty list.)

The correctness of Adjust can be shown by:

1. noting that no adjustment is required if the unrestricted case assignments are all integers (line 1) since the unrestricted case quickest multipath end-to-end delay is a lower bound for the minimum restricted case end-to-end delay
2. showing that with the input determined by Fractional_Adjust, Integer_Adjust creates assignments that provide the multipath's minimum end-to-end delay of any integer assignments for σ

For the second item we must show that the final adjusted assignments are all integer assignments for the message length and that any other all integer assignments for the message length cannot reduce the multipath's end-to-end delay. Toward that end, we note that in Fractional_Adjust and Integer_Adjust $\tau(P_i)$ and $\hat{\tau}(P_i)$ are additions to P_i's end-to-end delay that would result from increasing P_i's segment assignment to some integer values above its unrestricted case quickest multipath assignment. In Fractional_Adjust $\tau(P_i)$, computed at line 1, is the addition from an assignment increase to the next higher integer. In Integer_Adjust for $P_i \in LIST$, $\tau(P_i)$ is the addition from assignment increases in Fractional_Adjust and Integer_Adjust (Fractional_Adjust lines 3 and 5 and Integer_Adjust lines 2 and 7) and $\hat{\tau}(P_i)$ is the sum of $\tau(P_i)$ and the end-to-end delay addition from another one unit increase in P_i's assignment (Integer_Adjust lines 2 and 7).

Algorithm **Fractional_Adjust**(*MP, L, τ, LIST, ΔT*)
1. $\forall\ P_i \in P,\ \tau(P_i) \leftarrow \frac{1-(L(P_i)-\lfloor L(P_i) \rfloor)}{R(P_i)}$;
2. $U \leftarrow \sum_{P_i \in P} \left((L(P_i) - \lfloor L(P_i) \rfloor \right)$; $\Delta T \leftarrow 0$;
3. Create an array *LIST*, of all $P_i \in P$ ordered by ascending $\tau(P_i)$;
4. $\forall\ P_i \in$ LIST **do**
5. **if** P_i is the *U*th or earlier element of *LIST* **then** $L(P_i) \leftarrow \lfloor L(P_i) \rfloor + 1$;
6. **else** $L(P_i) \leftarrow \lfloor L(P_i) \rfloor$;
7. Remove from *LIST* all P_i after the *U*th P_i in *LIST*;
8. **if** $LIST \neq \phi$ **then** $\Delta T \leftarrow \tau(P_k)$ where P_k is the last path in *LIST*;
9. return (*LIST, L, ΔT*);

Let $MP = (P, R) = ((P_1, P_2, ..., P_r), (R(P_1), R(P_2), ..., R(P_r)))$ with message assignments $LS(MP) = (LS(P_1), LS(P_2), ..., LS(P_r))$ be an unrestricted case quickest multipath for a message of integer length σ and let $W = \sum_{P_i \in P} \left(\lfloor L(P_i) \rfloor \right)$. If MP and LS are the inputs to Fractional_Adjust then $U = \sigma - W$, where $U = \sum_{P_i \in P} \left((L(P_i) - \lfloor L(P_i) \rfloor \right)$ (lines 1 and 2). Let $LF(P_i)$ be the value of $L(P_i)$ and $\overline{LIST}$ be $LIST$ on exit from the subroutine (where $LIST$ is created at line 3 and reduced at line 7). Then $LF(P_i) = \lfloor L(P_i) \rfloor$ if $P_i \notin \overline{LIST}$ and $LF(P_i) = \lfloor L(P_i) \rfloor + 1$ if $P_i \in \overline{LIST}$ (lines 5-7). Hence $\forall\ P_i \in P$, $LF(P_i)$ is a non-negative integer (since $LS(P_i) \geq 0$) and $\sum_{P_i \in P} LF(P_i) = \sum_{P_i \notin \overline{LIST}} LF(P_i) + \sum_{P_i \in \overline{LIST}} LF(P_i) = \sum_{P_i \notin \overline{LIST}} \lfloor LS(P_i) \rfloor + \sum_{P_i \in \overline{LIST}} \left(\lfloor LS(P_i) \rfloor + 1 \right)$ $= \sum_{P_i \in P} \lfloor LS(P_i) \rfloor + U = W + U = \sigma$. Therefore Fractional_Adjust modifies the segment assignments to be non-negative integer assignments for σ. Further ΔT is the increase in MP's end-to-end delay for σ from Fractional_Adjust's adjustments to the message assignments. This is so because P_i's assignment will have been increased only if $P_i \in \overline{LIST}$, such an increase for P_i's assignment will have increased its end-to-end delay by $\tau(P_i)$, $\Delta T = \tau(P_k)$ where P_k is the last element of $\overline{LIST}$ and $\overline{LIST}$ is ordered by increasing $\tau(P_i)$ (see lines 3, and 5-8).

Let $LI(MP) = (LI(P_1), LI(P_2), ..., LI(P_r))$ be the MP integer assignments for σ created by Integer_Adjust. Any change made by Integer_Adjust to an $L(P_i)$ is made as a coupled change of +1 and -1 unit with an $L(P_l)$, $l \neq i$, (at lines 7 and 11). Hence the changed assignments remain integers and $\sum_{P_i \in P} LI(P_i) = \sum_{P_i \in P} LF(P_i)$ $= \sum_{P_i \in P} LS(P_i) = \sigma$. Further, changes are made only to $L(P_i)$'s for which $P_i \in \overline{LIST}$ and hence for which $L(P_i) \geq 1$ on entry into Integer_Adjust (Fractional_Adjust lines 4, 5 and 7). Since any $L(P_i)$ that is reduced is reduced only once (because its P_i is removed from $LIST$ at line 9 before the reduction) and each reduction is only a one unit reduction (line 11), therefore all assignments remain non-negative integers.

Algorithm **Integer_Adjust**(*MP, L, τ, LIST, ΔT*)

1. $\forall\ P_i \in LIST$ **do**
2. $\quad \hat{\tau}(P_i) \leftarrow \tau(P_i) + 1/R(P_i)$;
3. $LST \leftarrow LIST$; Restructure LST into a heap with element values $1/\hat{\tau}(P_i)$;
4. Restructure $LIST$ into a heap with element values $\tau(P_i)$;
5. **if** $LIST \neq \hat{\phi}$ **then** $P_k \leftarrow$ root of $LIST$; $Q \leftarrow$ root of LST;
6. **while** $LIST \neq \hat{\phi}$ and $\hat{\tau}(Q) < \Delta T$ **do**
7. $\quad L(Q) \leftarrow L(Q) + 1$; $\tau(Q) \leftarrow \hat{\tau}(Q)$; $\hat{\tau}(Q) \leftarrow \tau(Q) + 1/R(Q)$;
8. $\quad$ Restructure LST into heap with element values $1/\hat{\tau}(P_i)$;
9. $\quad$ Remove P_k from $LIST$;
10. $\quad$ Restructure $LIST$ into heap with element values $\tau(P_i)$;
11. $\quad L(P_k) \leftarrow L(P_k) - 1$;
12. $\quad P_k \leftarrow$ root of $LIST$; $\Delta T \leftarrow \tau(P_k)$; $Q \leftarrow$ root of LST;
13. return ($L, \Delta T$);

Let P_{rt} be the root of $LIST$ and Q_s be the root of LST at the beginning of an iteration of the line 7-12 loop. Then at the beginning of the iteration $\Delta T = \tau(P_{rt})$ (Fractional_Adjust lines 3 and 8 and Integer_Adjust lines 4,5,

10 and 12) and $\forall P_l \in LIST, \tau(P_l) \leq \tau(P_{rt})$[48]. Only $\tau(Q_s)$ is changed during the iteration[49] and at the end of the iteration $\tau(Q_s)$ is less than ΔT at the beginning of the iteration (lines 6 and 7). Therefore after P_{rt} is removed from *LIST* and *LIST* is restructured (lines 9 and 10), $\forall P_l \in LIST, \tau(P_l) \leq \tau(P_{rt})$. Hence, if P_{rnew} is the root of *LIST* after the restructuring, $\tau(P_{rnew}) \leq \tau(P_{rt})$. Since ΔT is set to $\tau(P_{rnew})$ after *LIST* is restructured (line 12), ΔT either is decreased or remains the same as a result of the iteration. Therefore ΔT is never increased during Integer_Adjust.

Let $T = MP$'s unrestricted case end-to-end delay for σ, $X = \Delta T$ at the end of Fractional_Adjust and $Y = \Delta T$ and $\widetilde{LIST}$ be *LIST* at the end of Integer_Adjust. Then $Y \leq X$ and Y is the minimum value of ΔT since ΔT never increases during the execution of Integer_Adjust. Now let $L(P_i) = LI(P_i) + 1$ and $L(P_j) = LI(P_j) - 1$ for some $j \neq i$, where $1 \leq i, j \leq r$. Also $\forall k, 1 \leq k \leq r$ and $k \neq i, j$ let $L(P_k) = LI(P_k)$. If $P_i \notin \overline{LIST}$ then $L(P_i) = \lfloor LS(P_i) \rfloor + 1$ and hence P_i's end-to-end path delay will be $T + \tau(P_i) \geq T + X \geq T + Y$ (Fractional_Adjust lines 3-8). If $P_i \in \overline{LIST}$ and $P_i \in \widetilde{LIST}$ then $L(P_i)$ will not have been reduced during execution of Integer_Adjust (since if it were reduced, at line 11, it would previously have been removed from *LIST* at line 9). Therefore P_i's end-to-end delay will be $T + \hat{\tau}(P_i) \geq T + Y$, since any path in $\overline{LIST}$ is also in *LST* (line 3) and at the conclusion of Integer_Adjust no P_i in LST has $\hat{\tau}(P_i) <$ Y (lines 3, 5, 8, 12 and 6). Finally $P_i \in \overline{LIST}$ and $P_i \notin \widetilde{LIST}$ [50], then P_i's end-end delay will be $T + \tau(P_i) \geq T + Y$, since the addition of a message unit to $L(P_i)$ restores the message unit subtracted from it after it is removed from *LIST* (lines 9 and 11) and when P_i is removed from *LIST*, $\tau(P_i) = \Delta T \geq Y$. Hence, since in all cases the unit increase of $LI(P_i)$ and corresponding decrease in $LI(P_j)$ doesn't reduce *MP*'s end-to-end delay (i.e., cause *MP*'s end-to-end delay to be $< T + Y$), and since any change in *LI* that maintains the *MP* segment assignments as non-negative integers summing to σ requires at least one segment assignment be increased by at least one unit, *LI* minimizes *MP*'s end-to-end delay in the restricted case for σ.

Adjust's complexity depends upon the number of paths in the multipath *MP* and that number is $\leq m$, since non-restrictive case multipaths produced by MTMP have no more than m paths. Fractional_Adjust's line 1, 2 and line 5-6 loops have no more than m iterations and since each iteration has complexity O(1), the loops' complexities are O(m). The creation of LIST as an ordered array of m elements at line 3 has complexity O($m$log$m$) = O($m$log$n$) if the array is created by a heap sort algorithm, see Cormen et. al. (Cormen et. al. 1990) for heap operations. So Fractional_Adjust's complexity is O($m$log$n$).

Integer Adjust's line 2 and lines 7-12 loops have less than m iterations (the later loop because it removes a path from *LIST* in each iteration). Creating heap *LST* at line 3 has complexity O($m$log$m$) and restructuring *LIST* into a heap at line 4 has complexity O(m) since it is ordered by ascending $\tau(P_i)$ prior to line 4. Restructuring *LST*

---

[48] Here we are viewing the heap as consisting of paths where the values of the elements are the elements' $\tau(\cdot)$. Hence if *LIST*'s elements j, $2j$ and $2j + 1$ are respectively P_i, P_k, and P_l, then in conformance with the heap property, $\tau(P_i) \geq \tau(P_k)$ and $\tau(P_i) \geq \tau(P_l)$, see Cormen et. al. (Cormen e.t. al. 1990).

[49] The root of *LST* has the lowest value of $\hat{\tau}(\cdot)$ of any path in *LST* since a *LST* element value is its $1/\hat{\tau}(\cdot)$ (lines 3 and 8). Hence the iteration adds a unit to the segment assignment of the path whose end-to-end delay is increased least by such an addition (lines 5, 12 and 7).

[50] If $\overline{LIST} \neq \hat{\phi}$, then $\widetilde{LIST} \neq \hat{\phi}$. We shall prove this by showing that if, at the beginning of an iteration of the lines 7-12 loop, Q is the root of *LST*, then $Q \in LIST$ and $Q \neq P_k$, where P_k is the root of *LIST*. Since only P_k is removed from *LIST* during an iteration, this implies that *LIST* is never emptied. At the beginning of an iteration $Q \in LIST$, because i) *LS*T is a restructured copy of $\overline{LIST}$, i.e., *LIST* on entry to Integer_Adjust, (line 3) and ii) Q wasn't previously removed from *LIST* by Integer_Adjust. The latter is true since if Q were previously removed, then $\tau(Q)$ would have had to equal the value of ΔT at the time of the removal (Fractional_Adjust line 8 and Integer_Adjust lines 5, 12, 9), but $\tau(Q) < \hat{\tau}(Q) < \Delta T$ (lines 2, 7 and 6) and Integer_Adjust never increases ΔT or decreases $\tau(Q)$. Finally, since $\tau(Q) < \Delta T$ and $\Delta T = \tau(P_k)$, $Q \neq P_k$.

at line 8 can be accomplished with a "heapify" operation beginning at the root since the root path, Q, is the only path in *LST* that has undergone a change of its $\hat{\tau}(\cdot)$. Restructuring *LIST* at line 10 is required because of the change of $\tau(Q)$ and the removal of P_k. It can be accomplished by moving Q to the root of *LIST*[51], then, if Q hadn't been childless, moving what had been a childless decedent of Q to Q's old position and performing a heapify operation beginning at that position, and, finally, performing a heapify operation beginning at the root. Thus both restructurings in the lines 7-12 loop have the complexity of a heapify operation which is O(log$m$), hence the totality of loop iterations has complexity O($m$log$m$). Consequently, since an iteration of the line 2 loop has complexity O(1), Integer_Adjust's complexity is O($m$log$m$) = O($m$log$n$). Since Adjust's line 1 has complexity O(m) and Fractional_Adjust and Integer_Adjust have complexity O($m$log$n$), Adjust has complexity O($m$log$n$).

[51] Here we assume that an array of path positions in *LIST* ordered by path index is created when *LIST* is restructured into a heap and modified with each path movement within and from *LIST*. Creation and maintenance of this array allows finding Q's position in *LIST* in one O(1) operation and hence does not add to the complexity of the *LIST* restructuring.

APPENDIX D. COMPLEXITY OF GENERATING QUICKEST MULTIPATHS

D.1 FLOW DECOMPOSITION MTMP COMPLEXITY IMPACT

The multipath flow decompositions dominate the complexity of MTMP2 and determine the asymptotic complexity of MTMP1. Hence a reduction of the complexity of the MTMP operations for generating quickest multipaths would reduce MTMP's complexity or asymptotic complexity. We shall examine the potential for such a reduction by first considering the particular case where all link delays are positive. Then we shall consider generating the multipaths in a manner that doesn't involve flow decomposition.

D.2 NETWORKS WITH ONLY POSITIVE LINK DELAYS

Decompose_to_Path_Flows on page 20 in Subsection 5.2 that carries out the decompositions is constructed to deal with flow loops. However any network flow for which an MTMP flow decomposition takes place will have no loop flow on a link with a non-zero delay (see Lemma 2 in Section 3 on page 8). It is instructive to analyze the network of Figure 5 below on this page to see potential impacts of zero delay links. Here if D(*u, v*) = D(*v, u*) = 0, then MTMP will create multipath MP_1 = ([*s, v, u, t*], 10) as a quickest multipath for message lengths greater than 0 and less than or equal 40 (corresponding to end-to-end delays greater than 3 and less than or equal to 7). In maximizing the flow on the next N_f, it might choose flow along path [*s, u, v, t*] resulting in a flow of 10 along each network link and therefore a loop flow of 10 along loop [*u, v, u*]. A flow decomposition could then lead to the multipath MP_2 = (([*s, v, t*], [*s, u, t*]), (10, 10)) as a quickest multipath for message lengths greater than or equal 40 (corresponding to end-to-end delays greater than or equal to 7). However if *D(u, v) = D(u, v)* = 1, then, while MTMP will again first create multipath MP_1 (as a quickest multipath for messages greater than 0 and less than 20 corresponding to end-to-end delays from 4 to 6), it will next choose generalized path [[*s, u, v, t*][1, -1, 1]] to maximize the flow on the next N_f resulting in a new flow on *N* in which *f(u, v) = f(v, u)* = 0, i.e., there will be no loop flow (as Lemma 2 indicates). Note that the resulting quickest multipath for message lengths greater than or equal to 20 (corresponding to end-to-end delays of 6 or more) will be MP_2.

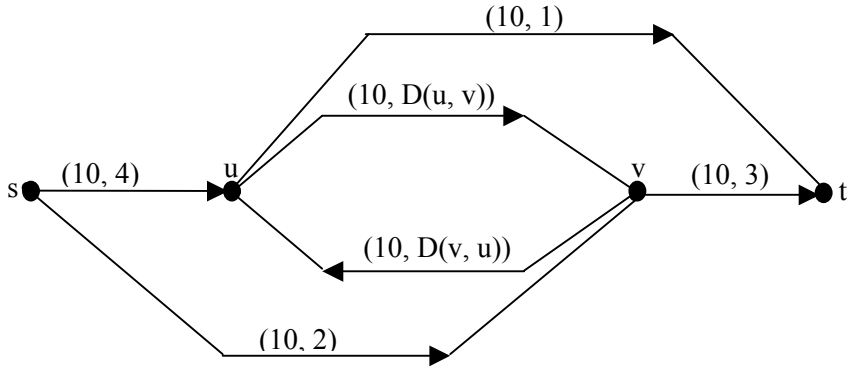

Fig. 5. Network illustrating augmenting flow choices.

If a network whose link capacities are positive integers has network flow *f** where *f** may contain flow loops, the upper bound on the number of iterations of Decompose_to_Path_Flows main loop is *m*, where *m - f**, iterations may lead to flow loop detections rather than path generations. However when all network link delays are greater than zero, the number of iterations is bounded above by *f**. In this case Decompose_to_Path_Flows' complexity is O(*nf**). Since:

1. *f** is bounded above by nC_{max}

2. there are less than nC_{max} decompositions in MTMP
3. the complexity of Decompose_to_Path_Flows' main loop is O(n)

therefore, the complexity of the totality of MTMP's decompositions is O($n^3C^2_{max}$). This complexity expression is neither superior nor inferior to O(n^2mC_{max}), the complexity expression we derived for the totality of the decompositions when loop flows are taken into consideration. Taking both complexity expressions into account, the complexity of the totality of the decompositions is O(n^2C_{max} (min(m, nC_{max}))) when all link delays are greater than zero. Hence the decompositions, even in this case, dominate all other aspects of MTMP2 (see Table 1 in Subsection 5.3 on page 21) as well as determining the asymptotic complexity of MTMP1.

D.3 MULTIPATH GENERATION ONE GENERALIZED PATH AT A TIME

We noted in Section 4 (on page 11) that if an MTMP flow maximization led to only positive link flows, i.e., positive $g(u, v)$, then a multipath created from the $g(u, v)$ could be "added" to the previous MTMP generated multipath to create the next MTMP multipath. Thus it might be fruitful to contemplate modifying the generic MTMP to combine the flow maximizations and multipath constructions. Each multipath could then be constructed by adding one flow augmenting generalized path at a time to the evolving multipath. Here if the flow augmenting generalized path is a path, then its addition has complexity O(n) and hence the path's selection that has complexity O(m), if done by a breadth first search, dominates its addition. Therefore, if all of the flow augmenting generalized paths generated by MTMP are paths, then MTMP's complexity will be determined by Min_Path's complexity. Hence, for a network with non-negative integer link delays and positive integer capacities, MTMP's complexity will be O($n(n$log$n + m)$min(C_{max}, D_{max})).

However we also have to be able to deal with flow augmenting generalized paths that have one or more counter flow links, i.e., cfl's. When incorporating such a generalized path into the existing multipath, the generalized path and multipath must be converted to a set of paths. Here the sum of the flow rates of the new set of paths must equal the sum of the multipath paths' flow rates plus the generalized path's flow rate. We present below on this page an algorithm, Supplement_Multipath, that does that. We assume for the subsequent discussion that all link capacities are positive integers.

Algorithm **Supplement_Multipath**(P_q, $R(P_q)$)

1. $CLIST \leftarrow$ list of P_q cfls in order of appearance in P_q;
2. $clength \leftarrow$ length of $CLIST$; $CP_s \leftarrow \widetilde{\phi}$; $\forall (u, v) \in E$, $PF((u, v)) \leftarrow 0$;
3. $\forall (u, v) \in CLIST$ **do**
4. $\quad PF((u, v)) \leftarrow R(P_q)$; $CP_u \leftarrow \widetilde{\phi}$; $CP_v \leftarrow \widetilde{\phi}$;
5. $i \leftarrow 1$;
6. **while** $i \leq clength$ **do**
7. $\quad (u, v) \leftarrow CLIST(i)$;
8. $\quad$ **while** $PF((u, v)) \neq 0$ **do**
9. $\quad\quad$ find P_j such that (u, v) is in P_j and $R(P_j) > 0$;
10. $\quad\quad$ $(CFF, PF) \leftarrow$ Create_Subpaths(P_j, $R(P_j)$, PF, P_q, CP);
11. $\quad\quad$ $R(P_j) \leftarrow R(P_j) - CFF$;
12. $\quad\quad$ **if** $R(P_j) = 0$ **then** Remove_from_Structures(P_j);
13. $\quad i \leftarrow i + 1$;
14. Create_New_Paths(P_q, $CLIST$, $clength$, $R(P_q)$, CP);

Supplement_Multipath's approach is an expansion of a technique that can be used to combine a simple generalized path P with a single cfl link (u, v) and a number of simple paths that contain link (u, v), where P and the paths have the same initial and final nodes, e.g., s and t. As an example of the technique's result,

suppose P's flow rate is FR and (u, v) is a link in paths Q^1 and Q^2 where each path has flow rate $\frac{2}{3} FR$. Let $P^1_s = P(s, v) + Q^1(v, t)$, $Q^1_s = Q^1(s, u) + P(u, t)$, $P^2_s = P(s, v) + Q^2(v, t)$ and $Q^2_s = Q^2(s, u) + P(u, t)$, where $P(s, v)$ is the subpath of P from s to v and $P(u, t)$, $Q^1(s, u)$, $Q^1(v, t)$, $Q^2(s, u)$ and $Q^2(v, t)$ are analogously defined. Then the paths P^1_s and Q^1_s each with flow rate $\frac{2}{3} FR$, P^2_s and Q^2_s each with flow rate $\frac{1}{3} FR$ and Q^2 with flow rate $\frac{1}{3} FR$ have the same total flow rate and data flow (for sufficiently large time) as P, Q^1 and Q^2 with their respective flow rates of FR, $\frac{2}{3} FR$ and $\frac{2}{3} FR$.

Operating in the context of an MTMP algorithm, Supplement_Multipath, starts with a multipath of r paths, $P_1, P_2, \ldots, P_r$ from s to t with flow rates $R(P_1), R(P_2), \ldots, R(P_r)$ and with a simple generalized path $P_q = (u_0, u_1, d_1), (u_1, u_2, d_2) \ldots, (u_{p-1}, u_p, d_p)$ with flow rate $R(P_q)$ where $p \geq 3$, $u_0 = s$ and $u_p = t$ (and $d_1 = d_p = 1$). P_q has between 1 and p -2 cfls and is a flow augmenting generalized path for the residual network created by the flow that results from the composition of the multipath's path flow rates. Hence there is sufficient forward flow in the multipath paths to "account for" the counter flow in the cfls of P_q. Supplement_Multipath deals with this more general case by identifying a set of multipath paths that account for the flow in P_q's cfls. It determines subpaths of these paths and subpath flow rates, storing the subpaths and their flow rates in a set, CP, of stacks (the CP value it passes to its subroutines is a pointer to CP stack pointers). Then it joins those subpaths and subpaths of P_q to create a subset of a new set of paths where the new set's flow rates sum to the sum of the multipath and P_q's flow rates. Here the multipath is initially presented to the Supplement_Multipath in a set of structures (that will be discussed in the next subsection) that facilitate the algorithm operation.

Supplement_Multipath first executes an initialization procedure in which it creates a list of the generalized path's, i.e., P_q's, cfls, sets the "unaccounted for" counter flow in each of P_q's cfls to P_q's flow rate of $R(P_q)$ and creates a set of empty subpath stacks (lines 1 - 4). Here $PF((u, v))$ is the unaccounted for counter flow in (u, v), and C_v is a stack for subpaths and flow rates where the subpaths originate at v. Then starting from the first cfl in P_q (line 5) the algorithm progresses through all the cfls moving to a succeeding cfl only after it has accounted for all the counter flow in a "current" cfl (lines 8, 13 and 7). It identifies one or more multipath paths for each cfl that together account for the remaining unaccounted for counter flow in the link (line 8 and 9). It, via a call to Create_Subpaths (line 10), creates a set of subpaths of each such path, then appropriately reduces the path's flow rate and, via calls to Remove_from_Structures (not shown here), removes the path from the structures if its flow rate has been reduced to zero (lines 10-12).

Create_Subpaths, shown on page D-4, uses the path P_j and path flow rate $R(P_j)$ passed to it to develop a set of subpaths that account for as much of the remaining counter flow in P_q's cfls as possible. It begins its execution by creating a list, $FLIST$, of its links that are cfls in P_q with remaining unaccounted for counter flow (line 1). Then it successively accounts for the counter flow in a decreasing set of cfls in each iteration of its main loop (lines 4 - 11). At each iteration it first determines a flow rate FP that will either fully account for the remaining counter flow in the cfl in $FLIST$ with the minimum unaccounted for counter flow or will utilize the remaining unused P_j flow (line 4). Then it creates the subpaths of P_j from s to the first link in $FLIST$, between links in $FLIST$ and from the last link in $FLIST$ to t and it reduces by FP the unaccounted for counter flow in each link in $FLIST$, i.e., it reduces $PF((w, x))$ for each (w, x) in $FLIST$, (lines 5 - 9). At the end of each iteration it recreates $FLIST$, where this recreation will have at least one less cfl than the previous $FLIST$, reduces the remaining unused flow in P_j, i.e., UFF, and increases the consumed P_j flow, i.e., CFF (lines 10-11).

When Supplement_Multipath exits its main loop, the result of its operation is:

1. for each (w, x) that is a cfl in P_q:
    a) CP_x contains subpaths originating at x whose flow rates sum to $R(P_q)$
    b) the CP_v contain among them subpaths terminating at w whose flow rates sum to $R(P_q)$

2. the CP_s contains subpaths originating at s whose flow rates sum to the sum of the multipath path flow rate reductions made by the algorithm
3. the CP_v, $v \neq s$ contain among them subpaths terminating at t whose flow rates sum to the multipath path flow rate reductions made by the algorithm

---

Algorithm **Create_Subpaths**(P_j, $R(P_j)$, PF, P_q, CP)

1. $FLIST \leftarrow$ list of P_q cfls (w, x) in P_j with $PF((w, x)) > 0$ ordered by their link order in P_j;
2. $flength \leftarrow$ length of $FLIST$; $UFF \leftarrow R(P_j)$; $CFF \leftarrow 0$;
3. **while** $flength \neq 0$ and $UFF > 0$ **do**
4. $\overline{PF} \leftarrow \min_{(w,x) \in FLIST} PF(w,x)$; $FP \leftarrow \min(UFF, \overline{PF})$;
5. $(w, x) \leftarrow FLIST(1)$; Push (P_j, s, w, FP) onto CP_s; $(w, x) \leftarrow (y, z)$; $j \leftarrow 1$;
6. **while** $j < flength$ **do**
7. $(w, x) \leftarrow FLIST(j)$; $(y, z) \leftarrow FLIST(j + 1)$;
8. Push (P_j, x, y, FP) onto CP_x; $PF((w, x)) \leftarrow PF((w, x)) - FP$; $j \leftarrow j + 1$;
9. Push (P_j, z, t, FP) onto CP_z; $PF(y, z)) \leftarrow PF((y, z)) - FP$;
10. $FLIST \leftarrow$ list of P_q cfls (u, v) in P_j with $PF((u, v)) > 0$ ordered by their link order in P_j; $flength \leftarrow$ length of $FLIST$;
11. $UFF \leftarrow UFF - FP$; $CFF \leftarrow CFF + FP$;
12. return (CFF, PF, CP);

Given the situation described above, it follows that subpaths of CP and subpaths of P_q provide a total flow rate of $R(P_q)$ into and out of each P_q intermediate node (maintaining the required flow rate balance without resort to counter flow in P_q's counter flow links). Also the flow rates of the CP subpaths from s and to t exactly compensate for the reduction in flow rates of the original multipath paths. Therefore, via a call to Create_New_Paths, shown below on this page, Supplement_Multipath "splices" the CP subpaths together with subpaths of P_q to create a set of paths from s to t whose flow rates sum to $R(P_q)$ plus the sum to the multipath flow rate reductions made by the algorithm.

Algorithm **Create_New_Paths**(P_q, $CLIST$, $clength$, $R(P_q)$, CP)

1. $(u, v) \leftarrow CLIST(1)$; Push $(P_q, s, v, R(P_q))$ onto CP_s;
2. $(w, x) \leftarrow CLIST(clength)$; Push $(P_q, w, t, R(P_q))$ onto CP_w; $k \leftarrow 2$;
3. **while** $k \leq clength$ **do**
4. $(w, x) \leftarrow CLIST(k)$; Push$(P_q, u, x, R(P_q))$ onto CP_u; $(u, v) \leftarrow (w, x)$;
5. $k \leftarrow k + 1$;
6. $\forall$ cfls (u, v) in P_q, order CP_u and CP_v in ascending order of element flow rates;
7. **while** $CP_s \neq \tilde{\phi}$ **do**
8. ($LIST$, $length$, FR) $\leftarrow$ Determine_Path(CP);
9. $(P, u, v, FP) \leftarrow LIST(1)$; $FP \leftarrow FP - FR$;
10. **if** $FP > 0$ **then** top of $CP_s \leftarrow (P, s, v, FP)$;
11. **else** Pop CP_s;
12. $k \leftarrow 2$;
13. **while** $k \leq length$ **do**
14. $(P, u, v, FP) \leftarrow LIST(k)$; $FP \leftarrow FP - FR$;
15. **if** $FP > 0$ **then** top of $CP_v \leftarrow (P, u, v, FP)$;
16. **else** Pop CP_v;
17. $P \leftarrow$ Expand($LIST$); $P \leftarrow$ Untangle(P); Enter (P, FR) into structures;
18. return;

Create_New_Paths, after it performs its initialization steps, operates somewhat like the algorithm Decompose_to_Path_Flows on page 20 in subsection 5.2. However Create_New_Paths deals with subpaths rather than links. In its initialization steps it determines subpaths of P_q that consist of only ffls, assigns them the flow rate $R(P_q)$ and pushes them onto appropriate *CP* stacks (lines 1- 5). These subpaths connect *s* to the first cfl of P_q, each cfl to its subsequent P_q cfl and the last cfl of P_q to *t*. Create_New_Paths also orders the subpaths in the CP stacks (line 6) by descending flow rate (to possibly reduce the number of paths generated). Then its main loop (lines 8 - 17), via calls to Determine_Path, shown below, successively connects subpaths to construct paths from *s* to *t* (line 8). Determine_Path returns an ordered list, *LIST*, of subpaths of a constructed path and the path's flow rate *FR*. Here each *CP* subpath in *LIST* is the top of its stack.

Algorithm **Determine_Path**(*CP, PQ,* P_q);
1. *length* ← 0; *l* ← 0; ∀ *v* ∈ *V*, γ(*v*) ← 0; *y* ← *s*;
2. **while** *y* ≠ *t* **do**
3. (*PP, y, z, RR*) ← Top of CP_y;
4. **while** *z* = *y* **do**
5. Pop CP_y; (*PP, y, z, RR*) ← Top of CP_y;
6. *length* ← *length* + 1; *LIST*(*length*) ← (*PP, y, z, RR*); $\overline{RR}$ (*length*) ← *RR*;
7. **if** *z* = *t* **then** *y* ← *t*;
8. **elsif** γ(*z*) ≠ 0 **then** *l* ← γ(*z*);
9. **else** γ(*y*) ← *length*; *y* ← *z*;
10. **if** *l* ≠ 0 **then**
11. $FP \leftarrow \min_{l \le i \le length} \overline{RR}(i)$; *i* ← *l*;
12. **while** *i* ≤ *length* **do**
13. (*P, w, x, RR*) ← *LIST*(*i*); *RR* ← *RR* - *FP*;
14. **if** *RR* ≠ 0 **then** top CP_w ← (*P, w, x, RR*);
15. **else** Pop CP_w
16. γ(*x*) ← 0;
17. *length* ← *l* - 1; *l* ← 0; *y* ← *z*;
18. $FR \leftarrow \min_{1 < i \le length} \overline{RR}(i)$;
19. return(*LIST, length, FR*);

On return from Determine_Path, Create_New_Paths' main loop reduces the flow rate of each *CP* subpath in *LIST* by the path flow rate (lines 9 and 10 and 14 and 15) or pops the stack of the subpath if its flow rate equals the path flow rate, thereby removing the subpath from the stack, (lines 11 and 16). It then, via Expand (which is not shown here), expands the subpaths of *LIST* into a node sequence representation of the constructed path, removes, via Untangle (not shown here), any remaining path loops and enters, via Enter (not shown here), the path and its flow rate into the structures (line 17). Note that Determine_Path detects via γ(·) indicators (line 8) and eliminates (lines 11-17) "subpath loops". It eliminates such a loop by reducing the flow rates of subpaths in the loop by the minimum loop subpath flow rate, i.e., the loop flow rate, and removing from the *CP* stacks any loop subpath whose flow rate equaled the loop flow rate.

Create_New_Paths will generate a null subpath, i.e., a subpath that has the same starting and ending node and no links, when two P_q cfls are consecutive links in P_q. Create_Subpaths will also generate a null subpath if P_j has consecutive links that are P_q cfls that both have remaining unaccounted for counter flow. Determine_Path deals with these subpaths by removing those it encounters from the *CP* stacks (lines 4 and 5).

Each initial and final node of the subpaths used by Create_New_Paths is in P_q and, as noted above, these subpaths initially provide total flow rates of $R(P_q)$ into and out of each P_q cfl node. Each subtraction of flow into such a node is accompanied by a subtraction of flow out of the node. Thus the flow balance at the nodes is maintained and hence Determine_Path will never reach an intermediate node and not have a subpath available from that intermediate node. Each construction attempt succeeds in reaching t since "subpath loops" are eliminated during the attempt and only a finite number of subpaths exist (so the routine can't loop indefinitely). Further each path construction leads to the reduction of a CP_s subpath flow rate by at least one unit and a CP_s subpath whose flow rate would be reduced to zero is removed from CP_s (lines 9-11 of Create_New_Paths).

The paths constructed by Determine_Path will have a total flow rate equal to the sum of the flow rates of subpaths in CP_s at the initiation of the subroutine, since Create_New_Paths will not terminate until CP_s is empty (line 4). At the initiation, this sum is $R(P_q)$ plus the total of the flow rates removed from the original multipath paths. Hence the new multipath in the structures at the conclusion of Supplement_Multipaths has a flow rate equal to the sum of the original multipath and $R(P_q)$. Further, since Supplement_Multipath never increases the flow on a link, the new multipath provides at least the same data flow for times equal to or greater than the delay of P_q as the original multipath plus P_q[52] (see equations (17) and (19) in Section 3 on page 7). However since Supplement_Multipath is being run in the context of an MTMP, the original multipath and the original multipath plus P_q both provide maximum data flow for an end-to-end delay equal to $\tilde{D}(P_q)$, the delay of P_q. Hence the new multipath provides the same data flow as the original multipath and P_q for times equal to or greater than $\tilde{D}(P_q)$ and per Lemma 2 in Section 3 on page 8 has no path with delay > $\tilde{D}(P_q)$. Therefore at the end of Create_New_Paths' execution, the required new multipath is in the structures (i.e., Supplement_Multipath is correct)[53].

D.4 COMPLEXITY OF SUPPLEMENT_MULTIPATH AND RELATED OPERATIONS

The following set of structures is utilized by Supplement_Multipath and its subroutines:

1. S1, a matrix with $m + 2$ columns and $(n-1)C_{max}$ rows.
2. S2, a two row $m + 1$ column matrix
3. S3, an n column matrix containing paths from s to t
4. S4, another n column matrix where each row is related to the path in the corresponding row of S3
5. S5, a five column array where each row is related to the path in the corresponding row of S3 and where the first column element is a flow rate, second column element an indicator/pointer, the third column a pointer to an S1 row and the fourth and fifth columns pointers of a linked list of non-zero flow rate rows (though the first row will always be in the list even if it has a zero flow rate).

The primary purpose of the first two structures is to allow the rapid determination of a multipath path containing a cfl of the generalized path. The elements of the matrix are pairs that in the first m +1 columns consist of a preceding row pointer and a succeeding row pointer. The last column's elements consist of a row number and a NILL. The first m columns of S1 correspond to the links of the network. The pointers in the first m columns form a linked list of paths containing the column's link. The next to last column's pointers form a linked list of unused S1 rows and the last column of an S1 row contains the row number of the S3 path corresponding to the S1 row. The second structure's columns correspond to the first m + 1 columns of S1. The first row contains the head S1 row number and the second row the tail S1 row number of the column's

[52] Here we are assuming for convenience of exposition that data can be transmitted over P_q though it cannot actually be due to P_q's cfls.

[53] A variation in the multipath construction approach of this subsection collects all the generated flow augmenting paths for a residual network, i.e., an N_f. Then it enters all the paths from this set of generalized paths into the structures and applies a generalized Supplement_Multipath approach to the subset of generated generalized paths that have cfls, collectively accounting for all the counter flow in this subset. Here *CLIST* contains the cfls of all the generalized paths of the subset.

linked list. S1 is initialized by MTMP so that its first m columns and last column are all NILL pairs and its next to last column forms a circular list of all the matrix's rows.

The path P_j determined at line 9 of Supplement_Multipath (in Subsection D.3 on page D-2) is found by selecting the head row of the S1 column corresponding to the cfl link *(u, v)* then finding the path's S3 row number in the last column of this S1 head row. Here the S1 head row is found from the corresponding S2 link column so that determining P_j is an O(1) complexity operation.

The third structure, S3, as noted above, contains paths. Each non NILL element of S3 is a pair consisting of a node (i.e., a node number) and the link number of the path's link to the node (the link number corresponds to the link's S1 column and the first element has a NILL link number). The first $\tilde{n}$ elements of a row are node/link numbers and the last $n - \tilde{n}$ elements are NILL pairs (where the $\tilde{n}$ nodes are unique, $2 \leq \tilde{n} \leq n$ and $\tilde{n}$ varies from row to row). S4's columns correspond to network nodes and an S4 entry is the path position of the column's node in the corresponding S3 path (if the node is in the path). S4 is used by Expand in line 17 of Create_New_Paths (in Subsection D.3 on page D-4) to develop the sequence of nodes of P. Locating the first node of the subpath in S3 is, through use of S4, an O(1) operation[54].

The flow rates of the last structure, S5, complete the definition of a multipath. The S5 indicator/pointers do not effect the operation of Supplement_Multipath. They are set to "new" by the algorithm any time their corresponding path is newly entered into S3 (for a purpose we shall go into after the algorithm discussion).

The subroutine Enter at line 17 of Create_New_Paths always appends a row to the S3, S4 and S5 when saving a new path in these structures. S1 however is never expanded beyond $(n - 1)C_{max}$ rows, where $(n - 1)C_{max}$ is an upper bound for the maximum number of paths in a quickest multipath.

S1 is the only structure initialized prior to other operations of an MTMP algorithm that uses Supplement_Multipath and, since S1 has $m+2$ columns, the complexity of such an initialization is $O(nmC_{max})$. An addition to the structures made by Enter, at line 17 of Create_New_Paths, occurs as follows:

1. the row number for the last row used in the S3, S4, and S5 structures is found from a length variable maintained by the algorithms, the variable is incremented, the path is entered into S3 and its node locations are entered into S4 at the new last row of these structures
2. the head row of the unused S1 row linked list is found from S2 and entered into the S5 row for the path along with the path flow rate, a "new" indicator and links to the previous tail and the head of the "non-zero flow rate path linked list" (with the previous tail "succeeding" and the head "preceding" values being modified to point to the path's row)
3. link entries are added in the path's S1 row in the columns corresponding to the path's links and each such column's head and previous tail entries are modified to make the path's row the tail of the column's linked list (yielding $< 3n$ entry changes in S1)
4. head and tail entries of the linked list of unused S1 rows (in S1 column $m + 1$) are changed, to reflect that the path's S1 row is in use, and the S3, S4, S5 row number of the path is entered into the last column of the path's S1 row
5. entries in the S2 columns corresponding to the path's links and the S2 column corresponding to the unused S1 rows are changed to reflect the changes to S1

A removal of a path from the structures is carried out by Remove_from_Structures at line 12 of Supplement_Multipath as follows:

[54] When Create_New_Paths constructs *CLIST* it also constructs an array for P_q that is analogous to an S4 row so the start of a P_q subpath can be similarly found in an O(1) operation.

1. the flow rate in the path's S5 row is set to zero and, if the path's row isn't the first S5 row, its preceding row's and succeeding row's entries are modified so the path's row is removed from the linked list of non-zero flow paths
2. the path's link entries in its S1 row (whose row number is found from the path's S5 third column entry) are set to NILLS and appropriate changes are made to what had been its preceding row's and succeeding row's entries to remove the path's row from the linked list of each column corresponding to a link of the path (requiring $< 3n$ S1 entry modifications)
3. the path's entry and the head and previous tail entries in the linked list of unused S1 rows is modified to make the path's row the tail of this linked list
4. entries in the S2 columns corresponding to the path's links and the S2 column corresponding to the unused S1 rows are changed to reflect the changes to S1

Each of the enumerated addition and removal steps can be carried out with complexity $O(n)$ (and some with a superior complexity). Hence Enter and Remove_from_Structures both have complexity $O(n)$.

The initialization steps of Supplement_Multipath (lines 1-5) have complexity $O(m)$ since the operation that sets all the links $PF((\cdot,\cdot))$ values to zero dominates the initialization. The number of Supplement_Multipath main loop iterations (lines 7-12) is $< n$ since there $< n$ cfls in P_q, and the number of inner loop (lines 9-13) iterations is $< nR(P_q)$, since each inner loop iteration reduces a path flow rate by at least one unit and the total reduction of flow rate is no more than $R(P_q)$ units per cfl of P_q. The main loop instructions exclusive of the inner loop have complexity O(1) hence the complexity of the totality of the main loop iterations exclusive of its inner loop is $O(n)$. Create_Subpaths initialization steps (lines 1 and 2), see page D-4 in Subsection D.3, have complexity $O(n)$ since determining if a P_j link should be in *FLIST* requires only checking if its $PF((\cdot,\cdot))$ value is greater than zero. Thus, since the choice of P_j (at line 9) is an O(1) operation and the complexity of Remove_from_Structures (called at line 12) is $O(n)$, the complexity of the totality of iterations of Supplement_Multipath's inner loop exclusive of Create_Subpaths' main loop (lines 4-11) is $O(n^2R(P_q))$.

Each iteration of Create_Subpaths' main loop and each iteration of its inner loop (lines 7-8) accounts for at least one unit of counter flow and therefore the number of iterations of either loop is $< nR(P_q)$. The only operations within these loops that aren't O(1) operations are the *FP* calculation at line 4 and the *FLIS*T regeneration at line 11 both of which have complexity $O(n)$. Therefore the complexity of the totality of Create_Subpaths' loop iterations is $O(n^2R(P_q))$.

Create_New_Paths' initialization (lines 1-6), see page D-4, is dominated by the reordering of the *CP* stacks that has complexity $O(nR(P_q))$[55]. Each iteration of Create_New_Paths main loop (lines 8-17) creates a path and reduces the flow rate of a C_s subpath by at least one unit (lines 9-11). Since the total of the C_s stack flow rates is $<$ (# of P_q cfls + 1) $R(P_q)$, the number of iterations, and hence paths created, $< nR(P_q)$. Each iteration of the main loop's interior loop (lines 14-16) reduces the flow rate in a *CP* subpath by at least one unit (lines 14-16). Since the total flow rates of the *CP* subpaths is $\leq 3(n-3)R(P_q) + R(P_q)$ (a maximum of $3R(P_q)$ for each cfl link of P_q, i.e., $R(P_q)$ each for the set of created path subpaths into and from each cfl and the P_q subpath from the cfl, and $R(P_q)$ for the P_q subpath from s to the first P_q cfl) the number of each loop's iterations is $< 3nR(P_q)$.

[55] The sum of the flow rates of each nonempty stack's entries is $R(P_q)$ (or $2R(P_q)$ if the stack's index is a node between two counter flow links) except for CP_s where the sum is $< nR(P_q)$. Hence the length of the nonempty stacks is $O(R(P_q))$ except CP_s's length that is $O(nR(P_q))$. In ordering the stacks, the algorithm determines the unique flow rates of stack entries. Then it orders the unique flow rates, copies the stack entries to arrays for each flow rate and then copies them back into the stack in flow rate order. The complexity of the ordering operation is $O((\textit{number of unique flow rates})^2)$ and the maximum number of unique flow rates is less than the square root of the length of the stack. Hence for stacks other than CP_s the ordering's complexity is $O(R(P_q))$ and that is also the complexity of the other operations. Since there are less than n stacks other than CP_s and the ordering complexity of CP_s is $O(nR(P_q))$, the total ordering of *CP*'s stacks has complexity $O(nR(P_q))$).

Therefore, since all operations in the loops except subroutine calls are O(1) operations and Determine_Path's initialization (line 1), see page D-5 in Subsection D.3, is dominated by the initialization of the $\gamma(\cdot)$ whose complexity is O(n), the complexity of the totality of the executions of Create_New_Paths' loops, exclusive of Determine_Path's post initialization operations and Create_New_Paths' calls to Expand, Untangle and Enter (line 17), is O($n^2R(P_q)$).

Each iteration of Determine_Path's main loop (lines 3-17) is associated with the flow rate reduction of at least one unit of a CP_s subpath (by Create_New_Paths at lines 9-11). Further each iteration of the main loop's inner loops (line 5 and lines 13-16) causes either the discarding of a *CP* subpath or the reduction of a *CP* subpath's flow rate by at least one unit. Therefore the number of iterations of the main loop $< nR(P_q)$ and the number of iterations of each of its inner loops is $< 3nR(P_q)$. Since all of these loops' operations except the loop and path flow rate computations (lines 11 and 18) are O(1) operations[56], the complexity of the totality of the main loop and interior loop iterations, exclusive of the loop and path flow rate computations, is O($nR(P_q)$). Further the number of elements over which a minimization in a flow rate computation is made is $\leq$ the number of iterations in a corresponding loop execution. The corresponding loop execution is the succeeding execution of the line 13-16 interior loop in the case of the loop flow rate computation (line 11) and the preceding execution of the main loop in the case of the path flow rate computation (line 18). Therefore the totality of these computations has complexity O($nR(P_q)$).

The totality of executions of Expand and Untangle has a complexity of O*(total number of nodes in created paths*). Let *NF* = the sum over the nodes of the total flow into all the nodes of the subpaths in the *CP* stacks. Then $NF < (\#\ of\ nodes\ in\ P_q)R(P_q) + \sum_{paths} \Delta R(P_j)(\#of\ nodes\ in\ P_j)$ where *paths* is the set of paths used to account for P_q's counter flow and $\Delta R(P_j)$ is the resulting decrease in P_j's flow rate. Then $NF < nR(P_q) + n \sum_{paths} \Delta R(P_j) \leq nR(P_q) + n(\#of\ cfls\ in\ P_q)R(P_q) < nR(P_q) + n^2R(P_q)$. Hence, since any path generated by Determine_Path has a flow rate of at least 1, the total number of nodes in all the created paths $< nR(P_q) + n^2R(P_q)$. Therefore the complexity of the totality of the executions of Expand and Untangle is O($n^2R(P_q)$). Since there are $< n(R(P_q)$ new paths and Enter has complexity O(n), the totality of the executions of Enter is also O($n^2R(P_q)$). Hence, since all portions of Supplement_Multipath have complexity O($n^2R(P_q)$) (and some a superior complexity), Supplement_Multipath's complexity is O($n^2R(P_q)$).

Every time that the last augmenting generalized path for an MTMP generated N_f is processed, the multipath paths in S3 along with their S5 flow rates must be saved. Here the S5 columns defining a linked list of non-zero flow rate paths allows each multipath path to be identified in an O(1) operation[57]. The direct copying of all such MTMP multipaths has complexity O($n^3C_{max}{}^2$) (since there are $< nC_{max}$ multipaths each having $< nC_{max}$ paths and copying a path has complexity O(n)).

The complexity of saving paths can be reduced by maintaining a list, S6, of all multipath paths and saving S5 flow rates and pointers rather than the multipath paths themselves. Each time a new path is added to the structures, its S5 indicator/pointer is set to "new". Then after a complete set of paths for a residual network, a N_f, has been determined, those paths with a S5 "new" indicator are appended to S6 (which begins as a null array) and the S5 indicators are

[56] Supplement_Multipath, when it creates empty subpath stacks at line 1, also creates a temporary array indexed by node number. Each node, that is the start of a subpath, has as its entry in the array a pointer to the stack for subpaths starting at that node. Therefore any required stack can be found in an O(1) operation.

[57] The first S5 row is always the head of the list though it may contain a zero flow rate. Therefore its flow rate must be checked to determine if it should be included in the multipath. However all other paths in the list always have non-zero flow rates.

changed to pointers to their respective S6 paths. The complexity of copying the multipath paths to S6 and saving the MTMP produced path table multipaths is, with this approach, $O(n^2C_{max}(n + C_{max}))$. (Here less than n^2C_{max} paths are produced by all MTMP's flow maximization and multipath generations. This together with the complexity of a copy to S6 being $O(n)$ yields the first complexity term. Also less than $n^2(C_{max})^2$ pointers and flow rates need be copied to save multipaths, since there are less than nC_{max} multipaths each having less than nC_{max} paths. Hence we have the second term.) Therefore the MTMP complexity is $\underline{O(n^2C_{max}(n + C_{max}))}$, since the complexity of the totality of all other MTMP operations is superior to $O(n^3C_{max})$. If in addition to all link capacities being positive integers, all link delays are non-negative integers then, since there are less than nD_{max} multipaths, the MTMP complexity[58] is $\underline{O(n^2C_{max}(n + \min(C_{max}, D_{max}))}$.

D.5 DUPLICATE MULTIPATH PATHS

A disadvantage of the previous section's Supplement_Multipath approach is that it, unlike the flow decomposition approach, may generate multipaths that contain duplicate paths[59]. For example, in the network of Figure 1 in Section 2 on page 5, the multipath for message lengths greater than or equal to 40 ($T \geq 7$) that would result from this approach is (([*s, u, t*], [*s, v, t*], [*s, u, t*], [*s, v, t*]), (8, 8, 2, 2)), in which both [*s, u, t*] and [*s, v, t*] appear twice. If we choose to remove duplicates after each N_f set of paths has been generated, then we can compare paths in S3 before copying any to S6. If we find a set of duplicate paths, then we can set the flow rate of one of the duplicate paths in S5 to the sum of all the duplicate path flow rates and set all of the other duplicate path flow rates to zero (removing those paths whose flow rates were set to zero from the linked list of non-zero flow rate paths maintained in S5). The complexity of this approach for all saved MTMP multipaths is $O(n^4(C_{max})^3)$ (since there are $< nC_{max}$ multipaths and $<n^2(C_{max})^2$ pair wise path comparisons per multipath and a comparison has complexity $O(n)$). Therefore, if this approach is used, it will apparently cause the altered MTMP's complexity to be inferior to the MTMP complexity when flow decomposition is used.

We shall conclude this appendix by considering an approach that eliminates duplicate paths without increasing the MTMP complexity. It does this through use of three additional structures and an added S5 column. These structures are:

1. S7, a n column array of paths from s to t and subpaths of paths from s to t
2. S8, a vector of pointers linking paths in S7 to paths in S3 (and rows of S4 and S5)
3. S9, a set of n column vectors in which each vector position represents a node

S7 is the main structure for relatively efficient avoidance of duplicate paths in multipaths. An S7 element is a triplet, containing a path node, an indicator, and an index. The indicators and indices join sets of rows at particular columns. Here joined rows all have the same subpath in the columns preceding the column at which the join is made but different nodes in the joined column, see Figure 6 on page D-11 (in which only the node portion of each entry is indicated in the main portion of the figure but the full triplets for four entries are displayed to the side of the main portion). An indicator takes one of three values: unjoined, base, or non-base. If the value is base, the index is to an S9 vector, i.e., a "join vector". If the value is non-base, the index is the row number of an S7 element in the join column that contains a base indicator. An S9 vector element value is either an S7 row number in which the corresponding node can be found in the join column or "Nill". A Nill indicates that the node can't be found at the join column in a row "joined" to the element whose index is to this S9 vector. An S8 element is the row number in S3 (and S4 and S5) of the path in the corresponding S7

[58] The complexity of an MTMP using the variation described in the footnote 53 on page D-6 is the same.

[59] Duplication will not occur when flow decomposition is used, because for each path that is generated one or more of the path's links are removed from further consideration and therefore cannot be part of any subsequent path that is generated for the multipath.

row. It will be Nill if the S7 row isn't a complete path from *s* to *t*. The added column in S5 contains S7 row numbers. The entry for an S5 row is the S7 row of the path (which is also in S3) corresponding to the S5 row.

An S7 element that has a base indicator is a "base" and one with a non-base indicator is a "non-base". A base and the set of non-bases whose indices are the base are a join (examples of joins are the connected circled elements of Figure 6). The base of a join is always in the earliest row of the join and each row other than the first row has one and only one non-base element. Also any row, other than the first, has a non-base at an earlier column than any bases in the row. The routine Determine_if_Duplicate_Path, shown on page D-12, determines if a path is in S7. It starts at the first row and first column and moves out along the row comparing S7 elements' nodes to the path's nodes (line 5). When a "non-match" is found, a check is made to see if the current element's row is joined at the current column to a row who's joined element's node is the path's node (line 6). If it isn't, then a "block" has occurred at the current row and column and the path isn't in S7 (line 7). If it is, the process continues at the joined row containing the path's node in the join's column (line 9). Either node *t* will be reached through this process or the process will be blocked indicating that the path isn't in S7. Each operation in the main loop (lines 5 - 14) has complexity O(1) and since the number of main loop iterations is < *n*, the routine's complexity is O(*n*)(here the S9 array allows the line 6 determination of whether the path node is in the array's join in an O(1) operation).

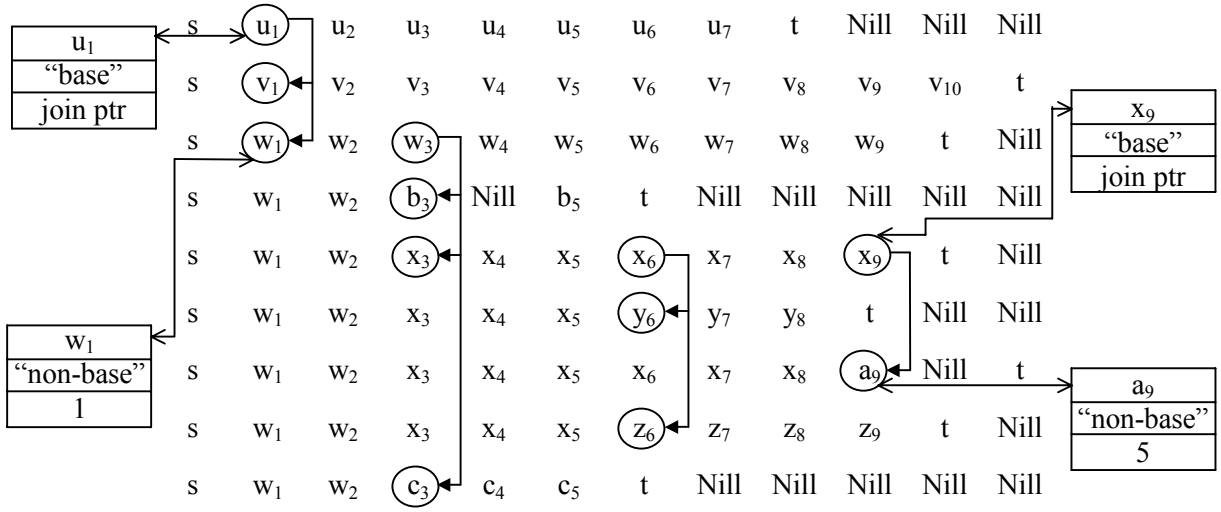

Fig. 6. S7 structure.

When a multipath path and its flow rate have been determined during an MTMP execution, a check is made for the path's existence in S7. If the path is in S7, then the path's flow rate in its S5 row is increased by the determined flow rate. Here the S8 entry for the path's S7 row contains the S5 row number for the path. Hence the flow rate increase requires a fixed number of O(1) operations. When a path flow rate is reduced to zero by Supplement_Multipath (line 11) then the path is removed from S7. The added column in S5 contains the S7 number row number of the path. Hence the path's S7 row can be found in an O(1) operation.

Addition of a path to S7 occurs only after an attempt to find the path in S7 has been blocked. If the block occurred at a Nill entry, the remaining subpath of the path is entered into the row in which the block occurred (followed by a Nill in the row if the path isn't *n* nodes long). Otherwise, the path is appended to S7. In either case, the S3 row number of the path is entered into the S8 position corresponding to the path's S7 row. When the path is appended a join must be altered or constructed. If the block occurs at an unjoined element, then the element at the block's row and column is converted to a base indexed to the next available S9 join vector (and the block's row number is entered into the position of the element's node in the join vector). Otherwise the

block is already a base. In either case, the path's S7 row number is inserted into the join's S9 vector at the position of the path's node (i.e., the node whose position corresponds to the block's column number). The indicator of the path's element in the block's column is set to non-base, and the element's index is set to the block's row (which is the join's base). Note that the complexity of an addition is O(n) since < n elements need to be entered into the path's S7 row, and the additional steps beyond the path entry are made up of a set of O(1) operations whose number is less than some fixed value[60].

```
Algorithm Determine_if_Duplicate_Path(P)
1.  row ← first row; column ← first column;
2.  element ← row and column's element; row_node ← element's node;
3.  path_node ← s; not_blocked ← 1; found = 0;
4.  while(found = 0 and not_blocked = 1) do
5.     if path_node ≠ row_node then
6.        if element not in join or join doesn't include element with path_node then
7.           not_blocked ← 0;
8.        else
9.           row ← row in join with element containing path_node
10.    if not_blocked = 1 and path_node ≠ t then
11.       column ← next column;
12.       element ← row and column's element; row_node ← element's node;
13.       path_node ← path P's next node;
14.    if (path_node = row_node = t) then found ← 1;
15. return(not_blocked, row, column)
```

The deletion of a path occurs as follows: The last column for which the path is joined is found. If its element's node isn't t, a Nill is placed in the triplet node position of the next column's element in the path's row. If the element's node is t, both the t in the element's node position and the path's row number in the join's S9 vector are overwritten with a Nill. Finally the S8 entry for the row is set to Nill. Note that the complexity of a deletion is O(n) since < n O(1) operations are required to find the last join in the path's S7 entry and the additional steps beyond finding the last join are made up of a set of O(1) operations whose number is less than some fixed value.

The maximum number of paths created by Supplement_Multipath in all of an MTMP execution is < n^2C_{max}. Thus checking all "new" paths and adding paths to and deleting them from S7 when required has complexity O(n^3C_{max}). Hence the S7 operations to deal with duplicate paths don't add to the complexity of Supplement_Multipath. Therefore the complexity of saving the multipaths will not increase.

S9 must be initialized with Nills (while S7 requires that only its first row and column's element be initialized with a Nill node). S9's size is limited by the maximum number of paths created for multipaths during an MTMP execution. Since the number of created paths is < n^2C_{max} and each S9 vector is n elements, the required number of S9 elements is < n^3C_{max}. Thus S9's initialization (with Nills) has complexity O(n^3C_{max}) and therefore doesn't increase Supplement_Multipath's complexity.

[60] The first path generated by MTMP is inserted into the first row of S7 and each path row element's indicator is set to non-joined. S8's indicator is set to 1 corresponding to the first row position of the path in S3, but no S9 operations are needed. A path that is subsequently added to S7 possibly, as noted above, additionally requires the creation of a base (which in turn requires the obtaining of a S9 join pointer and updating the next join pointer value) and requires adding its S7 row number to a S9 join vector.

The MTMP complexity of $O(n^2C_{max}\ (n + C_{max}))$, when the full process of adding one flow augmenting generalized path at a time to evolving multipaths, saving the multipaths at appropriate times and assuring that each set of multipath paths contain no duplicates is included, is neither superior nor inferior to the MTMP complexity of $O(n^2mC_{max})$ when flow decompositions are used. Further the corresponding one path at a time complexity of $O(n^2C_{max}\ (n + \min(C_{max}, D_{max}))$ when link delays are limited to be non-negative integers (as well as link capacities being limited to be positive integers) is neither superior nor inferior to MTMP1's complexity of $O(n^2((n^2/\log n) + m)\min(C_{max}, D_{max}))$ or MTMP2's complexity of $O(nm(C_{max} + n\min(C_{max}, D_{max})))$[61]. However readily derived upper bounds on the number of distinct paths for all the MTMP multipaths are n^2C_{max} for the "one generalized path addition at a time" approach (less than $nR(P_q)$ paths created for a flow augmenting path P_q with the sum of all the flow rates of the generated flow augmenting paths being less than nC_{max}) and nmC_{max} for the flow decomposition approach (less than nC_{max} multipaths each having no more than m paths since Decompose_to_Flows creates no more than m paths per decomposition). These upper bounds on the number of distinct paths for all the MTMP multipaths might be interpreted as an advantage from the "one generalized path addition to a time" approach.

[61] We have made algorithm comparisons in this report based only upon complexity measured in terms of execution time related bounds. However we recognize that other factors including memory requirement bounds and some practical aspects of algorithms may be at least of equal importance in determining which algorithm among a set of algorithms is "the best". In the case of the above approach to adding one generalized path at a time, the approach is neither inferior nor superior (from a complexity standpoint) to the flow decomposition approach described in the body of the report. However it is much more "computationally involved" in terms of the algorithms and structures that are used.

INTERNAL DISTRIBUTION

1. J. Barhen
2. T. Dunigan
3. W. Grimmell
4. J. Nichols
5. N. Rao
6. W. Wing
7. Q. Wu
8. Central Research Library
9. ORNL Laboratory Records-RC
10. ORNL Laboratory Records-OSTI

EXTERNAL DISTRIBUTION

11. Dr. Admela Jukan, Network Systems Cluster, Division of Computer and Network Systems, 4201 Wilson Boulevard, Suite 1175, Arlington, Virginia 22230

12. Dr. Sri Kumar, DARPA / IPTO, 3701 Fairfax Drive, Arlington, VA 22203-1714

13. Dr. Thomas D. Ndousse, Mathematical, Informational, and Computational Sciences Division, Germantown Bldg/SC-31, Office of Science, U.S. Department of Energy, 1000 Independence Avenue, SW, Washington DC 20858-1290

14. Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831

15. Professor Sridhar Radhakrishan, School of Computer Science, University of Oklahoma, Norman, OK 73019

16. Professor Guoliang Xue, Department of Computer Science/Engineering, Room 347 Goldwater Center, Arizona State University, Tempe, AZ 85287-5406