Network Upgrading Problems*

Doowon Paik

AT&T Bell Laboratories, Murray Hill, New Jersey 07974

Sartaj Sahni

Computer and Information Sciences Department, CSE 301, University of Florida, Gainesville, Florida 32611

Graphs with weights and delays associated with their edges and/or vertices are often used to model communication and signal flow networks. Network performance can be improved by upgrading the network vertices. Such an upgrade reduces the edge/vertex delays and comes at a cost. We study different formulations of this network performance improvement problem and show that these are *NP*-hard. We then consider one of the formulations and develop polynomial time algorithms for some special cases and pseudopolynomial time algorithms for others. *© 1995 John Wiley & Sons, Inc.*

1. INTRODUCTION

A communication network can be modeled as an undirected connected graph in which the edge delays (≥ 0) represent the time taken to communicate between a pair of vertices that are directly connected. Two vertices that are not directly connected can communicate by using a series of edges that form a path from one vertex to the other. The total delay along the communication path is the sum of the delays on each of the edges on the path. A shortest path is one for which the sum of the delays is the least. With respect to this undirected graph model, we define the following problems:

1.1. LinkDelay(x, δ)

In this problem, it is possible to upgrade each of the vertices in the undirected graph. If vertex v is upgraded, then the delay of each edge incident to v reduces by a factor x, $0 \leq x < 1$. So, if edge (v, w) has delay d before the upgrade, its delay is $x \times d$ following the upgrade. If both v and w are upgraded, its delay becomes $x^2 \times d$. The problem is to upgrade the smallest number of vertices so that following the upgrades no edge has delay $> \delta$.

1.2. ShortestPath(x, δ)

Upgrading a vertex has the same effect on edge delays as in LinkDelay(x, δ). This time, however, we seek to upgrade the smallest number of vertices so that following the upgrade there is no pair of vertices u and v for which the shortest path between them has delay $> \delta$.

1.3. Satellite(δ)

When a vertex is upgraded, a satellite up link and down link are placed there. Two vertices with satellite links can communicate in zero time. Let $dist(x, y)$ be the length of the shortest communication path between vertices x and y. Let CommTime(G) be $\max_{x,y \in V(G)}\{dist(x, y)\}$, where $V(G)$ is the set of vertices in G. The objective is to upgrade the smallest number of vertices so that CommTime$(G) \leq \delta$. Note that there is always a shortest communication path between two vertices that uses either

* This research was supported, in part, by the National Science Foundation under Grant MIP91-03379.

CCC 0028-3045/95/010045-14

0 or 2 satellite vertices (to use a satellite link, there must be a send and a receive vertex; further, there is no advantage to using more than one satellite link in any communication).

Signal flow in electronic circuits is often modeled by directed acyclic graphs (dags) [1, 3, 7]. Vertices represent circuit modules and directed edges represent signal flow. In a simplistic model, each has a delay of one. A module can be upgraded by replacing it with a functionally equivalent one using a superior technology. This reduces the delay of all edges incident to/from the module by a multiplicative factor x, $0 \leq x < 1$. In a simplistic model, this reduction factor is the same for all circuit modules. The cost of the upgrade is reflected in the weight associated with the vertex. Again, in a simplistic model, each vertex has unit weight (i.e., all vertices cost the same to upgrade). Since signals can travel along any of the paths of the dag, the performance of the circuit is governed by the length of the longest path in the dag. We wish to meet certain performance requirements by upgrading the fewest possible number of vertices. This is stated formally below:

1.4. LongestPath(x, δ)

Given a dag $G = (V, E)$ with positive edge delays, upgrade the smallest number of vertices so that the longest path in the upgraded graph has delay $\leq \delta$. When a vertex is upgraded, all edges incident to/from it have their delay changed by the multiplicative factor x.

In some networks, the primary delay may be the processing done in a vertex rather than the transmission via an edge. In such a case, we may associate a positive delay value with each vertex (rather than with each edge) and also a positive weight, which is the cost of upgrading the vertex. The delay of a path is the sum of the delays of the vertices on the path. When a vertex is upgraded, its delay is changed by a multiplicative factor x, $0 \leq x \leq 1$.

1.5. DVUP(x, δ)

In the *dag vertex upgrade problem* (DVUP), we are given a dag $G = (V, E)$ with positive vertex delays, $d(v)$, and positive weights, $w(v)$. We are to upgrade a minimum weight vertex set so that the longest path in the upgraded graph has delay $\leq \delta$. When a vertex v is upgraded, its delay becomes $x \times d(v)$.

Each of the five problems stated above is a simplified version of a more realistic problem. The more realistic problem may have different costs associated with the upgrade of different vertices and the upgrade factor may also vary from vertex to vertex. The complexity results obtained for the simpler models apply to the more realistic problems also.

Network performance enhancement through vertex upgrades has been considered before. Paik et al. [10] modeled the optimal placement of scan registers in a dag as a vertex splitting problem in a dag. The objective was to split the fewest number of vertices so that the resulting dag had no path of length $> \delta$. They showed that this problem was *NP*-hard for general dags but polynomially solvable for tree and series-parallel dags. Heuristics for this problem were proposed in [11]. In [12], Paik et al. showed that the problem of upgrading circuit modules so as to control signal loss can be modeled as a dag vertex deletion problem in which one seeks to delete the fewest number of vertices so that the resulting dag has no path of length $> \delta$. They showed that while this could be done in polynomial time for tree and series-parallel dags the problem was *NP*-hard for general dags.

DVUP(0, δ) was studied in [9]. They showed that

1. DVUP(0, δ) is *NP*-hard for directed chains.
2. DVUP(0, δ) can be solved in $O(n^3 \log n)$ time for unit weight unit delay dags (n is the number of vertices in the dag).
3. DVUP(0, δ) is *NP*-hard for dags with unit weight but nonunit delays when $\delta \geq 2$ and is solvable in $O(n^2)$ time when $\delta = 1$.

In addition, they proposed a backtracking algorithm and several heuristics for general dags.

Related vertex deletion problems were studied by Krishnamoorthy and Deo [5]. They showed that several interesting vertex deletion problems are *NP*-complete.

The first four problems defined above are considered in Sections 2–5, respectively. In these sections, we essentially obtain proofs of *NP*-hardness. These proofs make use of the following known *NP*-hard problems [2]:

1. Vertex Cover

Input: An undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Output: "Yes" iff there is a subset $V' \subseteq V$ with $|V'| \leq k$ such that for each edge $(u, v) \in E$ at least one of u and v belongs to V'.

2. Dominating Set

Input: An undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Output: "Yes" iff there is a subset $V' \subseteq V$ with $|V'| \leq k$ such that for $u \in V - V'$ there is a $v \in V'$ for which $(u, v) \in E$.

3. Maximum Clique

Input: A connected undirected graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Output: "Yes" iff there is a subset $V' \subseteq V$ with $|V'| \geq k$ such that every two vertices in V' are joined by an edge in E.

4. Exact Cover by 3-Sets (X3C)

Input: Set X with $|X| = 3q$ and a collection $C = \{C_1, C_2, \ldots, C_m\}$ of three element subsets of X such that $\cup_{i=1}^{m} C_i = X$.
Output: "Yes" iff C contains an exact cover for X, i.e., a subcollection $C' \subseteq C$ such that every element of X appears in exactly one member of C'.

In Sections 4–8, we study the DVUP$(0, \delta)$ problem for trees, series-parallel dags (SPDAGs), and general series-parallel dags (GSPDAGs), respectively. The results we obtain are

1. An $O(n)$ time algorithm for unit weight unit delay trees.
2. Pseudopolynomial time algorithms for trees, SPDAGs, and GSPDAGs with general weights and delays. These algorithms have complexity $O(n^3)$ when either $\delta = O(n)$, all weights are unit, or all delays are unit [note that if all delays are unit, then $d(G) \leq n$ and so $\delta < n$ for the problem to be nontrivial].

2. LinkDelay(x, δ)

When $\delta = 0$ and $x > 0$, LinkDelay(x, δ) can be solved in linear time. In case G has an edge with delay > 0, then the link costs cannot be made 0 by upgrading any subset of the vertices. If G has no edge with delay > 0, then no vertex needs to be upgraded. For all other combinations of δ and x, LinkDelay(x, δ) is *NP*-hard.

Theorem 1. LinkDelay(x, δ) is *NP*-hard whenever $\delta \neq 0$ or $x = 0$.

Proof. We shall use the vertex cover problem for this proof. Let $G = (V, E)$ be an instance of this problem. We obtain from G an instance G' of LinkDelay(x, δ) by associating a delay with each edge of G. If $\delta = 0$, this delay is one, and if $\delta > 0$, this delay is any number in the range $(\delta, \delta/x]$ [in case $x = 0$, the range is $(0, \infty)$]. Since $0 \leq x < 1$, upgrading vertex set A results in all links having a delay $\leq \delta$ iff A is a vertex cover of links in G' and, hence, of the edges in G. So, G' has an upgrading vertex set of size $\leq k$ iff G has a vertex cover of size $\leq k$. ■

3. ShortestPath(x, δ)

First, we show that ShortestPath$(0, 0)$ is *NP*-hard. We note that while at first glance Shortest-Path$(0, 0)$ appears to be identical to the vertex cover problem this is not so. Consider the graph of Figure 1. All edge delays are one. If the vertices $\{1, 3\}$ are upgraded, all shortest paths have a delay/cost of zero. However, $\{1, 3\}$ is not a vertex cover. ShortestPath$(0, 0)$ is also not the same as the dominating set problem in which one is seeking a vertex set A with the property that all vertices of the graph are either in A or adjacent to a vertex in A. Consider the graph of Figure 1 with edge $(2, 4)$ omitted. $\{1, 4\}$ is a dominating set of this graph. However, upgrading vertices 1 and 4 does not result in all shortest paths having a zero cost [the shortest path between 2 and 3 has cost equal to that of the edge $(2, 3)$ which might well be >0].

Lemma 1. ShortestPath$(0, 0)$ is *NP*-hard.

Proof. We shall use the exact cover by 3-sets problem for this proof. Let X, q, C, and m be as in the definition of X3C. Construct an instance $G = (V, E)$ of Shortest-Path$(0, 0)$ as below:

(a) G is a three-level graph with a root vertex r. This is the only vertex on level 1 of the graph.
(b) The root has $m + q + 2$ children labeled $C_1, C_2, \ldots, C_m, Z_1, Z_2, \ldots, Z_{q+2}$. These are the level 2 nodes of the graph. Child C_i represents set C_i, $1 \leq i \leq m$, while child Z_i is just a dummy node in G.
(c) The graph G has $3q$ nodes on level 3. These are labeled $1, 2, \ldots, 3q$. Node i represents element i of X, $1 \leq i \leq 3q$.
(d) Each node C_i on level 2 has edges to exactly three nodes on level 3. These are to the nodes that represent the members of C_i, $1 \leq i \leq m$.

An example of the construction is given in Figure 2. We shall show that the input X3C instance has answer "yes" iff the ShortestPath$(0, 0)$ instance G has an upgrade set of size $\leq q + 1$. First, suppose that the answer to the X3C instance is "yes." Then, there is a $C' \subseteq C$ such that

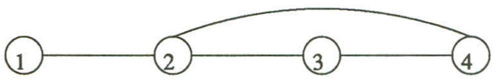

Fig. 1. An example.

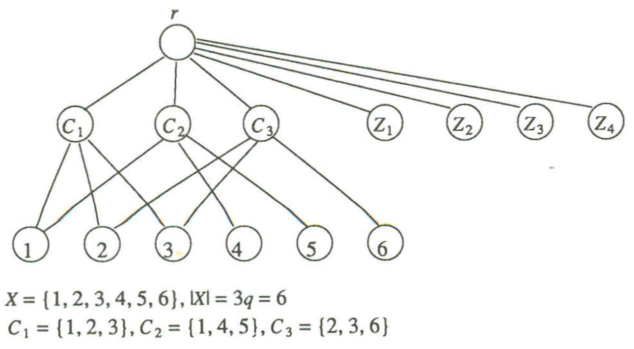
r
C_1
C_2
C_3
Z_1
Z_2
Z_3
Z_4
1
2
3
4
5
6
$X = \{1, 2, 3, 4, 5, 6\}, |X| = 3q = 6$
$C_1 = \{1, 2, 3\}, C_2 = \{1, 4, 5\}, C_3 = \{2, 3, 6\}$

Fig. 2. Construction of G for Lemma 1.

C' is an exact cover of X. Since $|X| = 3q$ and $|C_i| = 3$, $1 \leq i \leq m$, $|C'| = q$. Let $S = \{r\} \cup C'$. One may verify that S is an upgrade set for G and $|S| = q + 1$.

Next, suppose that G has an upgrade set S of size $\leq q + 1$. If $r \notin S$, then the shortest path from r to at least one of the Z_i's has length > 0 as at least one of the $q + 2$ Z_i's is not in S and every r to Z_i path must use the edge (r, Z_i). So, $r \in S$. When the vertices in S are upgraded, every vertex in G must have at least one zero length edge incident to it, as, otherwise, the shortest paths to it have length ≥ 1. In particular, this must be the case for all $3q$ level 3 vertices. Upgrading the root r does not result in any of these $3q$ vertices having a zero length edge incident to it. So, this is accomplished by the remaining $\leq q$ vertices in S. The only way this can be accomplished by an upgrade of $\leq q$ vertices is if these remaining vertices are a subset of $\{C_1, C_2, \ldots, C_m\}$ and this subset is an exact cover of X (this would, of course, require $|S| = q + 1$). So, $S - \{r\}$ is an exact cover of the input X3C instance. Hence, the X3C instance has output "yes" iff G has an upgrade set of size $\leq q + 1$. ■

Lemma 2. ShortestPath(0, 1) is *NP*-hard.

Proof. Once again, we use the X3C problem. The construction is similar to that of Lemma 1 except that we add the chain uvr to the root of G (see Fig. 3). If the answer to the X3C instance is "yes," then there is a subset C' that is an exact cover of X. Clearly, since $|X| = 3q$ and each C_i is of size three, $|C'| = q$. One may verify that if the vertices in $S = \{r\} \cup C'$ are upgraded then the shortest path between every pair of vertices has length ≤ 1. So, S is a vertex upgrade set for G. Hence, if the X3C instance has answer "yes," then G has an upgrade set of size $\leq q + 1$.

Next, suppose that G has an upgrade set S of size $\leq q + 1$. Suppose that $r \notin S$. Since $|S| \leq q + 1$ and there are $q + 2$ Z_i's, at least one Z_i is not in S. Let Z_j be such that $Z_j \notin S$. If there is another Z_k, $k \neq j$, $Z_k \notin S$, then the shortest path between Z_j and Z_k has length 2 following the upgrade. This violates the requirement that S is an upgrade set. So, there is no k, $k \neq j$, such that $Z_k \notin S$. This implies that $S = \{Z_1, Z_2, \ldots, Z_{q+2}\} - \{Z_j\}$, which, in turn, implies that the shortest path from Z_j to each of the vertices not in S has length ≥ 2. This again contradicts the assumption that S is an upgrade set for the constructed ShortestPath(0, 1) instance. So, $r \in S$.

Following the upgrade, each vertex in $\{1, 2, \ldots, 3q\}$ (i.e., each level 3 vertex) must have at least one zero length edge incident to it. To see this, first suppose there are two vertices, a, b, $a \neq b$, $a, b \in \{1, 2, \ldots, 3q\}$ that have no zero length edge incident to them. Since all paths between a and b include an edge incident to a and an edge incident to b and since there is no single edge incident to both a and b, the shortest path between a and b has length ≥ 2. This contradicts the assumption that S is an upgrade set for G. Now suppose there is exactly one level 3 vertex a that has no zero length edge incident to it following the upgrade. Then, each of the remaining $3q - 1$ level 3 vertices has at least one zero length edge incident to it. Each of the $\leq q$ vertices in $S - \{r\}$ causes 0, 1, or 3 level 3 vertices of G to have a zero length edge incident to it. So, to have $3q - 1$ level 3 vertices with zero length edges incident to them, we must have $|S - \{r\}| = q$ and $S - \{r\} \subseteq \{C_i, C_2, \ldots, C_m\}$ (only the C_i's result in three level 3 vertices having a zero length edge incident to them). But, then, the shortest path from u to a has length ≥ 2. So, there is no level 3 vertex a that has no zero length edge incident to it.

The above reasoning now implies that $S - \{r\} \subseteq \{C_1, C_2, \ldots, C_m\}$, $|S - \{r\}| = q$, and each of the $3q$ level 3 vertices has at least one zero length edge incident to it. Hence, $S - \{r\}$ is an exact cover of X. So, if G has an upgrade set of size $\leq q + 1$, then the answer to the X3C instance is "yes." ■

To show that ShortestPath(0, δ) is *NP*-hard for every δ, $\delta > 1$, we need to introduce a special graph that we call

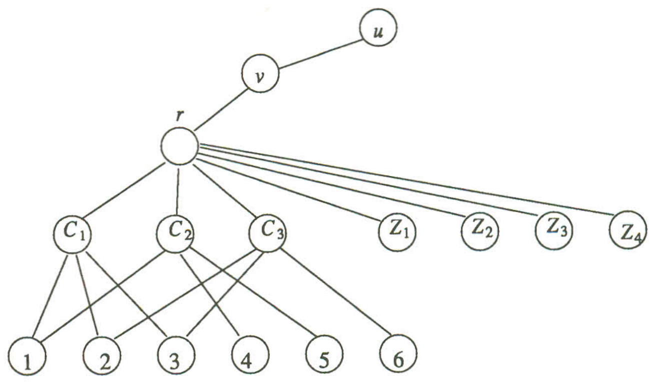

Fig. 3. Construction of G for Lemma 2.

a *daisy graph.* Let C_q, $q \geq 3$, denote the cycle graph with q vertices [Fig. 4(a)]. Assume that all edge delays are 1. All shortest paths in this graph have length $\leq \lfloor q/2 \rfloor$. Let $H(q, p)$ be the graph that results when p cycles C_q are joined at a common vertex v [Fig. 4(b)]. $H(q, p)$ is a *daisy graph* with p cycles of size q. Figure 4(c) shows the schematic for $H(q, p)$. All shortest paths in $H(q, p)$ have length $\leq 2\lfloor q/2 \rfloor$.

Let $H'(q, p)$ be the graph that results when vertex v of $H(q, p)$ is upgraded. All shortest paths in $H'(q, p)$ that end at vertex v have length $\leq \lfloor q/2 \rfloor - 1$. Consequently, all shortest paths in $H'(q, p)$ have length $\leq 2\lfloor q/2 \rfloor - 2$. To ensure that all shortest paths in $H'(q, p)$ that end at v have length $< \lfloor q/2 \rfloor - 1$, $q \geq 4$, it is necessary to upgrade at least $p + 1$ vertices (including v). To see this, note that at least two vertices in each C_q need to be upgraded for this and only one of these (i.e., v) can be common among the C_q's.

Lemma 3. ShortestPath$(0, \delta)$ is *NP*-hard for $\delta > 1$ and even.

Proof. Let $G = (V, E)$ be an instance of ShortestPath$(0, 0)$ in which all edge delays are 1. Construct G' by connecting a copy of $H(\delta + 2, n + 1)$, $n = |V|$, to each vertex u of G. This is done by adding an edge from the v vertex of $H(\delta + 2, n + 1)$ to vertex u (Fig. 5). This construction takes polynomial time in the size of G (we assume that δ is a constant).

We shall show that G has an upgrade vertex set A of size $\leq k \leq n$ [with respect to Shortest-Path$(0, 0)$] iff G' has one of size $\leq k + n$ [with respect to ShortestPath$(0, \delta)$]. Suppose that A is an upgrade vertex set for G and $|A| \leq k$. Then, $B = A \cup \{x | x$ is the v vertex of an $H(\delta + 2, n + 1)$ graph of $G'\}$ is an upgrade set for G'. To see this, observe that the length of the shortest path between all pairs of vertices i and j such that i and j are also vertices of G is 0 (as A is an upgrade set for G). If neither i nor j is a vertex of G, then the shortest path between them has length $\leq 2(\lfloor (\delta + 2)/2 \rfloor - 1) = \delta$ (note that when i and j are in different daisy graphs a path of length zero is used to go from the v vertex of one daisy graph to that of the other; also note that δ is even). Clearly, $|B| = |A| + n \leq k + n$.

Next, let B be an upgrade set for G' such that $|B| \leq k + n \leq 2n$. Let G'' be the graph that results when the vertices in B are upgraded. Let $A = B \cap V(G)$. $|A| \leq |B| - n \leq k$ as B must contain at least one vertex from each of the n daisy graphs (otherwise, there is at least one daisy graph which has a pair of vertices for which the shortest path has length $2\lfloor (\delta + 2)/2 \rfloor = \delta + 2$). If A is not a vertex upgrade set for G, then there is a pair of vertices i, j in G and, hence, G' for which the shortest path has length > 0 following the upgrade. Let l_i (l_j) be the length of the longest shortest path from a vertex in the daisy graph attached to i (j) to vertex i (j). Then, there is a shortest path in G'' with length $\geq l_i + l_j + 1$. Also, $l_i \geq \lfloor (\delta + 2)/2 \rfloor - 1 = \delta/2$ and $l_j \geq \delta/2$ [as to have l_i (l_j) $< \delta/2$ at least $n + 2$ vertices from the daisy graph attached to i (j) need to be upgraded; this is not possible, as, after accounting for the fact that B contains at least one vertex from each daisy graph, at most $k \leq n$ vertices remain while an additional $n + 1$ are needed]. So, G'' has a shortest path of length $\geq \delta + 1$, which contradicts the assumption that B is an upgrade set for G'. Hence, A is a vertex upgrade set for G. ■

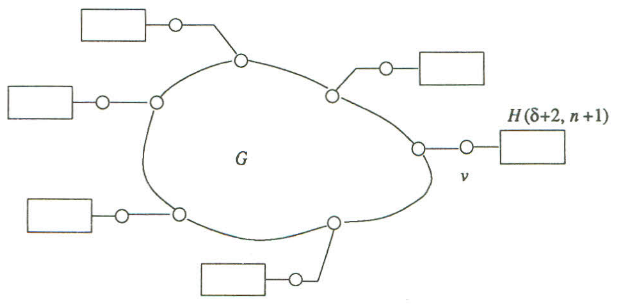

Fig. 5. Construction of G' from G in Lemma 3.

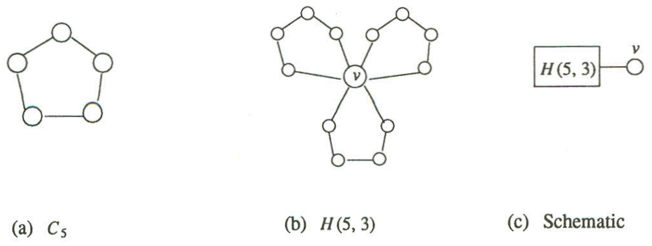

Fig. 4. Example cycle and daisy graphs.

Lemma 4. ShortestPath$(0, \delta)$ is *NP*-hard for $\delta > 0$ and odd.

Proof. Same as that of Lemma 3 except that we begin with an instance G of ShortestPath$(0, 1)$. The proof shows that G has a vertex upgrade set of size $\leq k$ (with respect to ShortestPath$(0, 1)$) iff G' has one of size $\leq k + n$ (with respect to ShortestPath$(0, \delta)$). ■

Lemma 5. ShortestPath(x, δ) is *NP*-hard for every pair (x, δ), $0 < x < 1$, $\delta > 0$.

Proof. Simply use the construction of Lemma 1. This time each edge of G of the form (r, Z_i) has delay $\delta/(2x)$ and the remaining edges have delay $\delta/(4x)$. ■

Theorem 2. ShortestPath(x, δ) is *NP*-hard whenever $x = 0$ or $\delta > 0$.

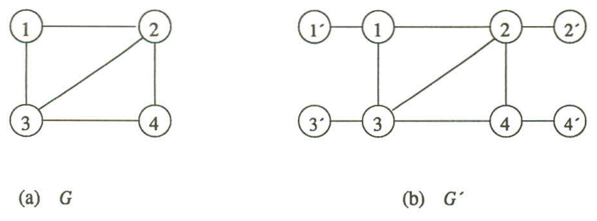

Fig. 6. Construction of Lemma 7.

Proofs. Follows from Lemmas 1–5. Note that even though Lemmas 1–4 only show that ShortestPath(0, δ) is *NP*-hard for integer δ, when δ is real, it may be replaced by $\delta' = \lfloor \delta \rfloor$ as the construction of Lemmas 1–4 use only unit delay edges. ■

4. Satellite(δ)

Satellite(0) is trivially solved. First, zero-length edges are eliminated by combining their end-points u and v into a single vertex uv. Remaining edges previously incident to u or v are now incident to uv. Duplicate edges are replaced by a single edge with the lower cost. If the resulting communication graph has at most one vertex, no satellite links are needed. When the number of vertices exceeds one, each vertex must be a satellite vertex. We shall show that Satellite(δ) is *NP*-hard for every δ, $\delta > 0$.

Lemma 6. Let G be a communication graph. Let S be the subset of vertices of G that are satellite vertices and let N be the remaining vertices (i.e., the nonsatellite vertices). CommTime(G) ≤ 1 iff the following are true:

(a) The vertices of N define a clique of G.
(b) Each vertex of N is adjacent to a vertex of S (i.e., S is a dominating set of G) unless $S = \varnothing$.

Proof. If N contains two vertices that are not adjacent in G, then the shortest path between them is of length at least two regardless of whether or not this shortest path utilizes satellites. So, the subgraph of G induced by the vertices of N is a clique. For (b), assume that $S \neq \varnothing$. If N contains a vertex v that is not adjacent to at least one satellite vertex, then v is at least distance two from every satellite vertex $s \in S$. Hence, CommTime(G) > 1. ■

Lemma 7. Satellite(1) is *NP*-hard.

Proof. We shall use the max clique problem for this proof. Let G be a connected undirected graph with n vertices. Let G' be the graph obtained by adding to G n edges of the type (i, i') where i is a vertex of G and i' is a new vertex (Fig. 6). The number of vertices in G' is $2n$. All edges of G' have unit delay. We claim that G has a clique of size $\geq k$, for any $k \geq 3$ iff G' has a vertex subset S, $|S| \leq 2n - k \leq 2n - 3$, such that by making the vertices of S satellite vertices CommTime(G') = 1. To see this, let A, $|A| \geq 3$, be any subset of vertices in G. If A forms a clique in G, then, by making all vertices of G' except those in A satellites, the communication time in G' becomes at most one as the vertices of A satisfy the conditions of Lemma 6 on nonsatellite vertices. So, if G has a clique of size $\geq k$, then G' has a satellite subset S of size $\leq 2n - k$. Next, suppose that G' has a satellite subset S of size $\leq 2n - k \leq 2n - 3$. Let N be the remaining vertices of G'. If N contains a vertex i' that is not in G, then from Lemma 6 it follows that $|N| \leq 2$ as the largest clique in G' that includes vertex i' has only two vertices in it (the other vertex being vertex i). In this case, $|S| \geq 2n - 2$. So, N contains no vertex that is not in G. Since N forms a clique in G' (Lemma 6), it forms a clique in G. Hence, if $|S| \leq 2n - k \leq 2n - 3$, G contains a clique of size $\geq k \geq 3$. While the *NP*-hard formulation of the max clique problem does not restrict k to be ≥ 3, it is easy to see that the problem remains *NP*-hard under this restriction. So, Satellite(1) is *NP*-hard. ■

For Satellite(2), we first observe that selecting the satellite vertices S so that S is a vertex cover or a dominating set results in CommTime(G) ≤ 2. However, the reverse is not true. For example, if G is a star graph, then CommTime(G) = 2 with $S = \varnothing$ [Fig. 7(a)], while $S = \varnothing$ is neither a vertex cover nor a dominating set for G. If G is the complete graph, K_n, on n vertices, then G has no vertex cover of size $< n - 1$. However, $S = \varnothing$ results in CommTime(G) = 1. Consider the graph G of Figure 7(b). $S = \{a, b\}$ results in CommTime(G) = 2. But, G has no dominating set of size < 3. So, there is no apparent direct connection between vertex sets S that result in CommTime(G) ≤ 2 and either vertex covers or dominating sets of G.

Lemma 8. Satellite(2) is *NP*-hard.

Proof. This proof utilizes the vertex cover problem. Specifically, we use the fact that the vertex cover problem

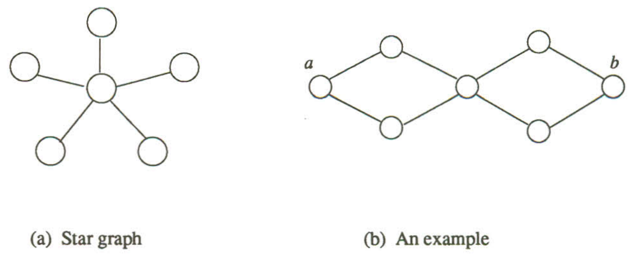

Fig. 7. Example graphs.

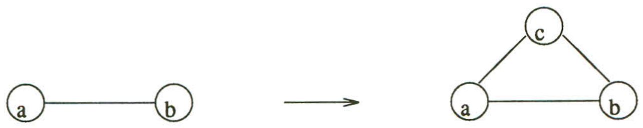

Fig. 8. Figure for Lemma 8.

limited to connected graphs of degree 5 and number of vertices ≥ 30 is *NP*-hard [2]. Let G be an instance of this problem. Assume that G has n vertices, $n \geq 30$, and that no vertex has degree > 5. Construct an instance G' of Satellite(2) by replacing each edge of G by the triangle graph of Figure 8.

We claim that G has a vertex cover of size $\leq k$ for any given $k < n$ iff G' has a vertex set S of size $\leq k$ such that making the vertices in S satellite vertices results in $\text{CommTime}(G') \leq 2$. As noted earlier, selecting S so that S is a vertex cover results in $\text{CommTime}(G) \leq 2$. Each vertex of G' that is not in G is unit distance from at least one vertex of S. Hence, making the vertices in S satellites results in $\text{CommTime}(G') \leq 2$. So, if G has a vertex cover of size $\leq k$, G' has a satellite set of size $\leq k$.

Next, let S be a satellite set for G' such that $|S| \leq k$. If, for every $(a, b) \in G$, at least one of a or b is in S, then S is a vertex cover of G. If, for every $(a, b) \in G$ such that $a \notin S$ and $b \notin S$, c [i.e., the new vertex introduced when triangulating (a, b)] is in S, replace c by a in S. The resulting S has the same size as the original S and is a vertex cover of G. If there is an $(a, b) \in G$ such that none of a, b, and c is in S, then $|S| \geq n$. To see this, let $A = V - \{u \mid u = a \text{ or } u = b \text{ or } u \text{ is adjacent to } a \text{ or } b\}$ where V is the set of vertices in G. Since no vertex of G has degree > 5, $|A| \geq n - 10$. Let B be the set of vertices not in G that are adjacent (in G') to the vertices of A. Since G may be assumed to be connected, each vertex of A has at least one edge incident to it. So, $|B| \geq (n - 10)/2$. Let $C = A \cup B$. $|C| \geq 1.5 \times (n - 10)$. Let w be a vertex in C. If $w \notin S$, then the shortest path from c to w has length ≥ 3. This is so, because if the path utilizes satellites, then c is at least 2 units from its nearest satellite and w is at least 1 units from its nearest satellite. If the path does not utilize satellites, then its length is ≥ 3 as w is not adjacent to either a or b. So, $C \subseteq S$ and $|S| \geq |C| \geq 1.5 \times (n - 10) \geq n$ for $n \geq 30$. This contradicts the assumption that $|S| \leq k < n$. So, the case when none of a, b, and c is in S does not arise. ■

Theorem 3. Satellite(δ) is *NP*-hard for $\delta \geq 1$.

Proof. For integer δ, the proof is by induction on δ. For the induction base, Lemmas 7 and 8 show that Satellite(1) and Satellite(2) are *NP*-hard. Assume that Satellite(δ) is *NP*-hard for δ, $1 \leq \delta \leq m$, where m is an arbitrary natural number greater than or equal to two. For $\delta = m + 1$, let G be an instance of Satellite($m - 1$). Let n, $n > 1$, be the number of vertices in G. Obtain an instance G' of Satellite($m + 1$) by attaching to each vertex of G the complete graph K_{n+1} on $n + 1$ vertices, $n > 1$ (Fig. 9). Each K_{n+1} has a vertex in common with G. All edge delays are one. G has a satellite set of size $\leq k < n$ [we may assume that $k < n$ as the case $k = n$ is trivially solvable in $O(1)$ time] iff G' has one of size $\leq k < n$. To see this, let S be a satellite set for G. Since the distance between any two vertices in K_{n+1} is one, S is a satellite set for G' (the distance between vertices of G' that are also vertices of G is $\leq m - 1$; between a vertex of G and one that is not in G, it is $\leq m$; and between two vertices not in G, it is $\leq m + 1$).

Next, let S' be a satellite set for G' such that $|S'| \leq k < n$. If S' contains only vertices that are in G, then S is a satellite set for G. To see this, note that if the shortest distance between two vertices x and y of G is $> m - 1$ after the vertices in S' have been made satellites, then the shortest distance in G' between a nonsatellite vertex $x' \neq x$ in the K_{n+1} attached to x and a nonsatellite vertex $y' \neq y$ in the K_{n+1} attached to y is $> m + 1$. Also, note that x' and y' must exist as $|S'| \leq k < n$ and a K_{n+1} has $n + 1$ vertices.

Suppose S' contains a vertex x that is not in G. x is a vertex of some K_{n+1}, say X. Let y be the vertex of this K_{n+1} that is also in G. Let $S'' = S' - \{x\} + \{y\}$. Clearly, $|S''| \leq |S'|$. Let G'_1 (G'_2) denote G' with the vertices in S' (S'') being satellite up link and down links. Let i and j be any two vertices of G'. The shortest path, P, between i and j in G'_1 has length $\leq m + 1$. If this path does not utilize the satellite link at x, then its length in G'_2 is also $\leq m + 1$. Suppose that the satellite link at x is used. If $j \neq x$, then change this path to use the satellite in G'_2 at y; i.e., if the old path is $P = i, \ldots, l, x, \ldots, j$, where l and x are the satellite vertices, the new path is $i, \ldots, l, y, j$ in case $j \in X$ and $i, \ldots, l, y, S(P)$ otherwise, where $S(P)$ is the suffix of P that goes from y to j. Such a suffix must exist as we may assume (l, x) is the only satellite link used on P and the only way to exit X without using a satellite is via vertex y. The length of the new path is $\leq$ that of the old one. If $j = x$, then consider a vertex z,

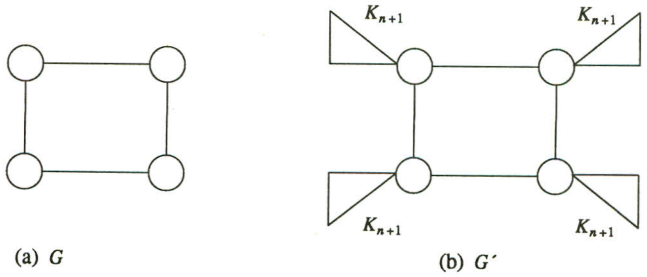

Fig. 9. Construction for Theorem 3.

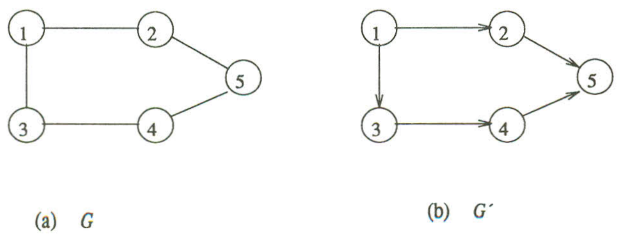

Fig. 10. Construction for Lemma 9.

$z \neq y$, of X that is not in S'. Such a z must exist as $|S'| \leq k < n$ and X has $n + 1$ vertices. The shortest path, $Q = i, \ldots, w, z$ in G'_1 from i to z has length $\leq m + 1$. If Q does not go through vertex x, then $Q' = i, \ldots, w, x$ is a path of length $\leq m + 1$ in G'_2. So, the shortest i to x path in G'_2 has length $\leq m + 1$. If Q goes through x, then the prefix of Q from i to x has length $< m + 1$ in G'_1. If this prefix does not use the satellite link at x, its length in G'_2 is also $<m + 1$. If it does, then by redirecting the satellite transmission to y and using the edge (nonsatellite link) from y to x, we get an i to x path in G'_2 that has length $\leq m + 1$. So, in all cases, moving the satellite links from x to y preserves the property that all shortest paths have length $\leq m + 1$.

By repeating this transformation on S' several times, we obtain a vertex set T such that $|T| \leq |S'| \leq k < n$, all vertices of T are in G, and T is a satellite set for G'. So, T is a satellite set for G.

When δ is a noninteger, replace δ by $\delta' = \lfloor \delta \rfloor$. Since our construction uses only unit delay edges, every solution for Satellite(δ) is also one for Satellite(δ') and vice versa. ■

5. LongestPath(x, δ)

Lemma 9. LongestPath(0, 0) is *NP*-hard.

Proof. Let G be a connected undirected graph. We shall construct an instance G' of Longest-Path(0, 0) such that G has a vertex cover of size $\leq k < n$ iff G' has a vertex upgrade set A' of size $\leq k < n$. To get G', orient the edges of G to begin at a lower index vertex; i.e., if (i, j), $i < j$, is an edge of G, then $\langle i, j \rangle$ is a directed edge of G' (Fig. 10). All edges of G' have unit delay. It is easy to see that G' is a dag and that A is a vertex cover of G iff $A' = A$ is a vertex upgrade set of the LongestPath(0, 0) instance G'. ■

To show that LongestPath(x, δ) is *NP*-hard for $x = 0$ and $\delta > 0$, we use the subgraph $J_{n,q}$ (Fig. 11) which is composed of n directed chains of size q that share a com-

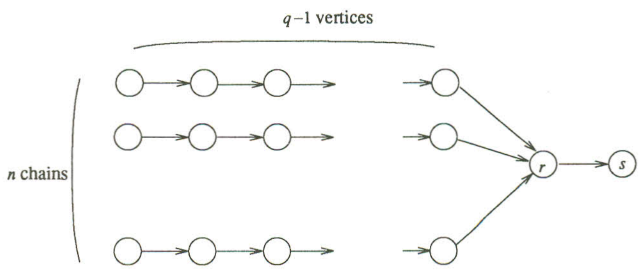

Fig. 11. $J_{n,q}$.

mon vertex r. This set of n chains is connected to a two-vertex-directed chain $\langle r, s \rangle$. Each edge of $J_{n,q}$ has unit delay. We see that the longest path in $J_{n,q}$ has length q, and that by upgrading the single vertex r, we can make the delay of this subgraph $q - 2$. However, to reduce the delay to $q - 3$, we need to upgrade at least $n + 1$ vertices.

Lemma 10. LongestPath(x, δ) is *NP*-hard for $\delta > 0$.

Proof. Let G be any instance of LongestPath(0, 0) in which all edge delays are one. A corresponding instance G' of LongestPath(x, δ) is obtained by attaching a copy of $J_{n,\lfloor \delta \rfloor+2}$ in case $x = 0$ and Q_n (Fig. 12) in case $x > 0$ to each vertex v of G that has in-degree zero. This is done by identifying the s vertex of $J_{n,\lfloor \delta \rfloor+2}$ with vertex v (i.e., these two vertices are the same). Note that n is the number of vertices in G. Let m be the number of vertices in G that have zero in-degree. One may verify that for any k, $k \leq n$, G has an upgrade set of size $\leq k$ iff G' has one of size $\leq m + k$. Hence, LongestPath(x, δ) is *NP*-hard for $\delta > 0$. ■

Theorem 4.

(a) Longest Path (x, δ) is *NP*-hard when $x = 0$ and $\delta = 0$ and also when $x \geq 0$ and $\delta > 0$.

(b) Longest Path (x, δ) is polynomially solvable when $x > 0$ and $\delta = 0$.

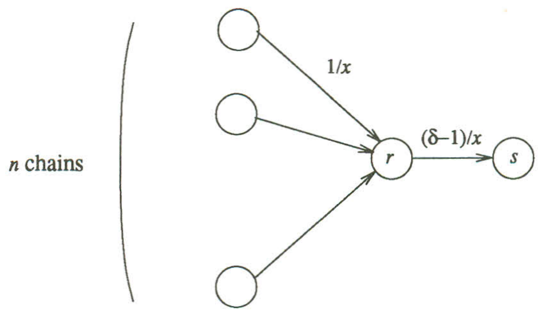

Fig. 12. Q_n.

Proof. (a) has been proved in Lemmas 9 and 10. For (b), if the dag has an edge with delay > 0, then it has no vertex upgrade set. If there is no such edge, then no vertex needs to be upgraded to ensure that the longest path has zero length. ■

6. DVUP(O, δ) FOR TREES

6.1. Rooted Trees with Unit Weight and Unit Delay

Let $d(G)$ be the length of the longest path in G. Let $G|X$ be the graph that results when the vertices in G are upgraded. Let $d(G|X)$ be the length of the longest path in $G|X$.

When the dag is a rooted tree T such that $w(v) = d(v) = 1$ for every vertex, the minimum weight vertex subset X such that $d(T|X) \leq \delta$ can be found in $O(n)$ time by computing the height, h, of each vertex as defined by

$$h(v) = \begin{cases} 1 & v \text{ is a leaf} \\ 1 + \max\{h(u) \mid u \text{ is a child of } v\}, & \text{otherwise} \end{cases}.$$

X is selected to be the set

$$X = \{v \mid h(v) > \delta\}.$$

The vertex heights can be computed in $O(n)$ time by a simple postorder traversal of the tree T [4]. The correctness of the procedure outlined above is easily established.

6.2. General Rooted Trees

Since a chain is a special case of a tree and since DVUP for chains with arbitrary weights and delays is known to be *NP*-hard [12], we do not expect to find a polynomial time algorithm for general trees. In this section, we develop a pseudopolynomial time algorithm (i.e., one whose complexity is polynomial in the number of vertices and the actual values of the vertex delays and weights). This algorithm has quadratic complexity when either all delays are unit or all weights are unit.

By establishing standard dynamic programming recurrences and solving these directly, one can obtain an algorithm with complexity $\Theta(\min\{\delta, \omega\} \times n)$, where $\omega = \sum_{u=1}^{n} w(u)$; i.e., the algorithm will take $c \times \min\{\delta, \omega\} \times n$ time on all trees with n vertices regardless of the values of the weights and delays. For this, we let $W(v, e)$ denote the minimum weight of an upgrading set for the subtree rooted at v and such that the upgraded subtree has delay less than or equal to e. It is easy to see that

$$W(v, e) = \begin{cases} w(v) + \sum_{x \in C(v)} W(x, e), & e \leq d(v) \\ \min\{\sum_{x \in C(v)} W(x, e - d(v)), w(v) \\ \quad + \sum_{x \in C(v)} W(x, e)\}, & d(v) < e \leq \delta, \end{cases}$$

where $C(v)$ represents the children of v. If r is the tree root, then $W(r, \delta)$ is the minimum weight of an upgrading set. $W(v, e)$ can be computed in $\Theta(\delta n)$ time as each vertex (other than the root) is the child of exactly one v. The upgrading set can also be found in this much time. In case $\delta > \omega$, we can use an alternate formulation. In this, we let $D(v, f)$ denote the minimum delay achievable for the subtree rooted at v by an upgrading set with weight at most f. The time to compute all the D's is $\Theta(\omega n)$.

We can arrive at an algorithm whose complexity is $O(\min\{\delta, \omega\} \times n)$ by using a tuple formulation rather than an array formulation. This algorithm has a run time that is bounded by $c \cdot \min\{\delta, \omega\} \times n$ for some constant c. However, depending on the actual weights and delays, this algorithm may take much less time.

For each vertex v, let $L(v)$ be a set of pairs (l, c) (l represents a delay and c a weight or cost), $l \leq \delta$, such that there is a weight c upgrading set for the subtree rooted at v that results in a delay of l (for the subtree). Let (l_1, c_1) and (l_2, c_2) be two different pairs such that $l_1 \leq l_2$ and $c_1 \leq c_2$. In this case, pair (l_1, c_1) *dominates* (l_2, c_2), i.e., pair (l_1, c_1) represents a preferred upgrading set. Let $S(v)$ be the subset of $L(v)$ that results from the deletion of all dominated pairs. If r is the root, then the least-cost pair in $S(r)$ represents the minimum weight upgrading set that results in a delay no more than δ.

We shall describe how to compute $S(r)$. Using the backtrace strategy of [4, cf. chapter on dynamic programming], we can compute the minimum weight upgrading set in $O(n)$ additional time. For a leaf vertex v, $S(v)$ is given by

$$S(v) = \begin{cases} \{(0, w(v))\}, & d(v) > \delta \\ \{(0, w(v)), (d(v), 0)\}, & d(v) \leq \delta. \end{cases}$$

For a nonleaf vertex v, $S(v)$ may be computed from the $S(u)$'s of its children $u_1, \ldots, u_{k_v}$. First, we compute $U(v)$ as the set of nondominated pairs of the form (l, c) where $l = \max\{l_1, l_2, \ldots, l_{k_v}\}$ and $c = \sum_{i=1}^{k_v} c_i$ for some set of pairs $(l_i, c_i) \in S(u_i)$, $1 \leq i \leq k_v$. Let $V(v)$ and $Y(v)$ be as below:

$$V(v) = \{(l, c + w(v)) \mid (l, c) \in U(v)\}$$

$$Y(v) = \{(l + d(v), c) \mid l + d(v) \leq \delta \text{ and } (l, c) \in U(v)\}.$$

Now, $S(v)$ is the set of nondominated pairs in $V(v) \cup Y(v)$. Since $S(v)$ contains only nondominated pairs, all pairs in $S(v)$ have different l and c values. So, $|S(v)| \leq \min\{\delta, \omega\}$, where $\omega = \sum_{u=1}^{n} w(u)$. Using the technique of [4], $S(v)$ can be computed from the $S(u)$'s of its children in time $O(\min\{\delta, \omega\} \times k_v)$. To compute $S(r)$, we need to compute $S(v)$ for all vertices v. The time needed for this is $O(\min\{\delta, \omega\} \times \sum k_v) = O(\min\{\delta, \omega\} \times n)$.

Note that for unit delay trees $\delta \leq n$, and for unit weight trees, $\omega = n$. So, in both of these cases, the procedure described above has complexity $O(n^2)$.

7. DVUP(O, δ) FOR SERIES-PARALLEL DAGS

A *series-parallel digraph,* SPDAG, may be defined recursively as

1. A directed chain is an SPDAG.
2. Let s_1 and t_1, respectively, be the source and sink vertices of one SPDAG G_1 and let s_2 and t_2 be these vertices for the SPDAG G_2. The parallel combination of G_1 and G_2, $G_1//G_2$, is obtained by identifying vertex s_1 with s_2 and vertex t_1, with t_2 [Fig. 13(c)]. $G_1//G_2$ is an SPDAG. We restrict G_1 and G_2 so that at most one of the edges $\langle s_1, t_1\rangle$, $\langle s_2, t_2\rangle$ is present. Otherwise, $G_1//G_2$ contains two copies of the same edge.
3. The series combination of G_1 and G_2, G_1, G_2, is obtained by identifying vertex t_1 with s_2 [Fig. 13(d)]. G_1G_2 is an SPDAG.

The strategy that we employ for SPDAGs is a generalization of that used in Section 6.2 for trees with general delays and weights. Let s and t, respectively, be the source and sink vertices of the SPDAG G. Let $D(l, Y, G)$ $(l \leq \delta)$, be a minimum weight vertex set that results in a delay of at most l. Y tells us whether or not the source and sink vertices are upgraded. Thus, there are four types of upgrading sets D:

(a) Sets which include neither the source nor sink. For these, $Y = \varnothing$.
(b) Sets which include the source but not the sink. These have $Y = \{s\}$.
(c) Sets which include the sink but not the source. For these, $Y = \{t\}$.
(d) Sets which include both the sink and the source. For these, $Y = \{s, t\}$.

With each upgrading set $D(l, Y, G)$, we associate a triple (l, c, Y) such that c is the weight of the upgrading set D. Let $f(G)$ denote the set of all of these triples for G.

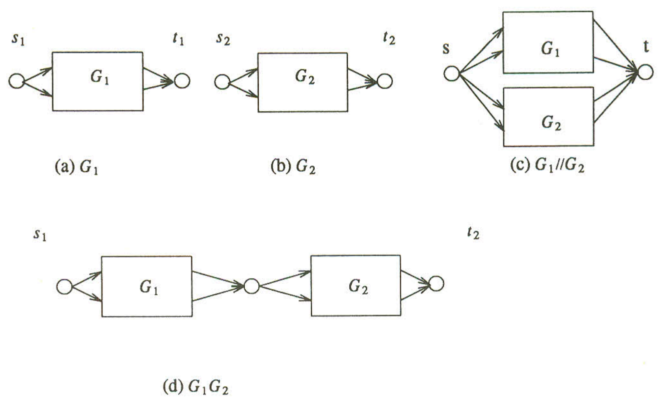

Fig. 13. Series-parallel digraph.

Let (l_1, c_1, Y_1) and (l_2, c_2, Y_2) be two different triples in $f(G)$. We shall say that triple (l_1, c_1, Y_1) *dominates* triple (l_2, c_2, Y_2) iff $l_1 \leq l_2$, $c_1 \leq c_2$ and $Y_1 = Y_2$. Let $F(G)$ be the set of triples obtained by deleting all dominated triples of $f(G)$. It is easy to see that the least-weight triple (i.e., the one with least c) in $F(G)$ corresponds to the minimum weight upgrading set for G.

As in the case for general trees, we shall show only how to compute $F(G)$. The minimum weight upgrading set may be obtained using a backtrace step as described in [4].

7.1. G is a Chain

Consider the case when G has only two vertices s and t. $F(G)$ is constructed using the code

```
$F(G) := \{(0, w(s) + w(t), \{s, t\})\}$
**if** $d(s) \leq \delta$, **then** $F(G) := F(G) \cup \{(d(s), w(t), \{t\})\}$
**if** $d(t) \leq \delta$, **then** $F(G) := F(G) \cup \{(d(t), w(s), \{s\})\}$
**if** $d(s) + d(t) \leq \delta$, **then** $F(G)$
    $:= F(G) \cup \{(d(s) + d(t), 0, \varnothing)\}$.
```

When G is a chain with more than two vertices, it may be regarded as the series composition of two smaller chains G_1 and G_2. In this case, $F(G)$ may be constructed from $F(G_1)$ and $F(G_2)$ using the algorithm to construct $F(G_1G_2)$ described in the next section.

7.2. G Is of the Form G_1G_2

The following lemma enables us to construct $F(G_1G_2)$ from $F(G_1)$ and $F(G_2)$.

Lemma 11. If $(l, c, Y) \in F(G_1G_2)$, then there is an $(l_1, c_1, Y_1) \in F(G_1)$ and an $(l_2, c_2, Y_2) \in F(G_2)$ such that

(a) $D(l_1, Y_1, G_1) = D(l, Y, G_1G_2) \cap V(G_1)$

(b) $D(l_2, Y_2, G_2) = D(l, Y, G_1G_2) \cap V(G_2)$

(c) $D(l, Y, G_1G_2) = D(l_1, Y_1, G_1) \cup D(l_2, Y_2, G_2)$

(d) $c = \sum_{u \in D(l,Y,G_1G_2)} w(u)$, $c_1 = \sum_{u \in D(l_1,Y_1,G_1)} w(u)$, $c_2 = \sum_{u \in D(l_2,Y_2,G_2)} w(u)$.

Proof. Let $A = D(l, Y, G_1G_2) \cap V(G_1)$ and $B = D(l, Y, G_1G_2) \cap V(G_2)$. Since $V(G_1G_2) = V(G_1) \cup V(G_2)$, $D(l, Y, G_1G_2) = A \cup B$. Let s_i and t_i, respectively, denote the source and sink vertices for G_i, $i \in \{1, 2\}$. s and t are the corresponding vertices for G_1G_2. We see that $s = s_1$, $t = t_2$, and $t_1 = s_2$.

CASE 1. $[t_1 \notin D(l, Y, G_1G_2)]$.

Let $l_1 = d(G_1|A)$, $Y_1 = A \cap \{s_1\}$, and $c_1 = \sum_{u \in A} w(u)$. We shall show that $(l_1, c_1, Y_1) \in F(G_1)$. Suppose it is not. Then, it must be dominated by a triple (l'_1, c'_1, Y_1) that is in $F(G_1)$. Let $C = D(l'_1, Y_1, G_1)$. It follows that $c'_1 = \sum_{u \in C} w(u)$ and that $(l'_1 + d(G_2|B) - d(s_2), c'_1 + \sum_{u \in B} w(u), Y)$ dominates $(l_1 + d(G_2|B) - d(s_2), c_1 + \sum_{u \in B} w(u), Y) = (l, c, Y)$. So, $(l, c, Y) \notin \mathrm{F}(G_1G_2)$. This contradicts the assumption on (l, c, Y). Hence, $(l_1, c_1, Y_1) \in F(G_1)$. Similarly, $(l_2, c_2, Y_2) \in F(G_2)$. (a)–(d) are an immediate consequence as $D(l_1, Y_1, G_1) = A$ and $D(l_2, Y_2, G_2) = B$.

CASE 2. $[t_1 \in D(l, Y, G_1G_2)]$

This is similar to case 1. ∎

Lemma 11 suggests the following approach to obtain $F(G_1G_2)$ from $F(G_1)$ and $F(G_2)$:

STEP 1. Construct a set Z of triples such that $F(G_1G_2) \subseteq Z$. This is obtained by combining together pairs of triples $(l_1, c_1, Y_1) \in F(G_1)$ and $(l_2, c_2, Y_2) \in F(G_2)$.

STEP 2. Eliminate from Z all triples that are dominated by at least one other triple of Z.

To avoid excessive complexity in the implementation of steps 1 and 2, we maintain $F(G)$ as four separate sets of triples. These are denoted by $F(G, Y)$, i.e., $F(G, Y)$ contains only those triples of $F(G)$ that correspond to upgrading sets that include the vertices in Y, where $Y \subseteq \{s, t\}$. We note that only the following triple combinations from $F(G_1)$ and $F(G_2)$ are permissible. s_i and t_i, respectively, denote the source and sink vertices of G_i $1 \le i \le 2$. Note that s_1 is also the source of G_1G_2 and t_2 is its sink.

1. Triple $(l_1, c_1, \emptyset)$ of $F(G_1, \emptyset)$ combines with a triple $(l_2, c_2, \emptyset)$ of $F(G_2, \emptyset)$ to produce a triple $(l_1 + l_2 - d(t_1), c_1 + c_2, \emptyset)$ of $F(G_1G_2, \emptyset)$.
2. Triple $(l_1, c_1, \emptyset)$ of $F(G_1, \emptyset)$ combines with a triple $(l_2, c_2, \{t_2\})$ of $F(G_2, \{t_2\})$ to produce a triple $(l_1 + l_2 - d(t_1), c_1 + c_2, \{t_2\})$ of $F(G_1G_2, \{t_2\})$.
3. Triple $(l_1, c_1, \{s_1\})$ of $F(G_1, \{s_1\})$ combines with a triple $(l_2, c_2, \emptyset)$ of $F(G_2, \emptyset)$ to produce a triple $(l_1 + l_2 - d(t_1), c_1 + c_2, \{s_1\})$ of $F(G_1G_2, \{s_1\})$.
4. Triple $(l_1, c_1, \{s_1\})$ of $F(G_1, \{s_1\})$ combines with a triple $(l_2, c_2, \{t_2\})$ of $F(G_2, \{t_2\})$ to produce a triple $(l_1 + l_2 - d(t_1), c_1 + c_2, \{s_1, t_2\})$ of $F(G_1G_2, \{s_1, t_2\})$.
5. Triple $(l_1, c_1, \{t_1\})$ of $F(G_1, \{t_1\})$ combines with a triple $(l_2, c_2, \{s_2\})$ of $F(G_2, \{s_2\})$ to produce a triple $(l_1 + l_2, c_1 + c_2 - w(t_1), \emptyset)$ of $F(G_1G_2, \emptyset)$.
6. Triple $(l_1, c_1, \{t_1\})$ of $F(G_1, \{t_1\})$ combines with a triple $(l_2, c_2, \{s_2, t_2\})$ of $F(G_2, \{s_2, t_2\})$ to produce a triple $(l_1 + l_2, c_1 + c_2 - w(t_1), \{t_2\})$ of $F(G_1G_2, \{t_2\})$.
7. Triple $(l_1, c_1, \{s_1, t_1\})$ of $F(G_1, \{s_1, t_1\})$ combines with a triple $(l_2, c_2, \{s_2\})$ of $F(G_2, \{s_2\})$ to produce a triple $(l_1 + l_2, c_1 + c_2 - w(t_1), \{s_1\})$ of $F(G_1G_2, \{s_1\})$.
8. Triple $(l_1, c_1, \{s_1, t_1\})$ of $F(G_1, \{s_1, t_1\})$ combines with a triple $(l_2, c_2, \{s_2, t_2\})$ of $F(G_2, \{s_2, t_2\})$ to produce a triple $(l_1 + l_2, c_1 + c_2 - w(t_1), \{s_1, t_2\})$ of $F(G_1G_2, \{s_1, t_2\})$.

If the triples in each set $F(G_i, Y)$ are stored in increasing order of delay (i.e., first coordinate), then since only nondominated triples are stored, they are also in decreasing order of weight (i.e., second coordinate). There is no need to store the third coordinate with each triple as triples with different third coordinates are stored separately. The triples for each of the above eight cases can be generated in time proportional to the product of the number of triples in each of the two sets being combined. When the triples from cases 1 and 5 have been generated, they may be merged together to obtain the triples of $F(G_1G_2, \emptyset)$, in ascending order. During this merge, dominated triples are eliminated. The time required for this merge is linear in the number of triples being merged. Triples from cases 2 and 6, cases 3 and 7, and cases 4 and 8 are also to be merged.

7.3. $G = G_1//G_2$

When $G = G_1//G_2$, we use Lemma 12 which is the analog of Lemma 11.

Lemma 12. If $(l, c, Y) \in F(G_1//G_2)$, then there is an $(l_1, c_1, Y_1) \in F(G_1)$ and an $(l_2, c_2, Y_2) \in F(G_2)$ such that

(a) $D(l_1, Y_1, G_1) = D(l, Y, G_1G_2) \cap V(G_1)$

(b) $D(l_2, Y_2, G_2) = D(l, Y, G_1G_2) \cap V(G_2)$
(c) $D(l, Y, G_1G_2) = D(l_1, Y_1, G_1) \cup D(l_2, Y_2, G_2)$
(d) $c = \sum_{u \in D(l,Y,G_1G_2)} w(u)$, $c_1 = \sum_{u \in D(l_1,Y_1,G_1)} w(u)$, $c_2 = \sum_{u \in D(l_2,Y_2,G_2)} w(u)$.

Proof. Similar to that of Lemma 11. ■

To obtain $F(G_1//G_2)$ from $F(G_1)$ and $F(G_2)$, we use the two-step approach used to compute $F(G_1G_2)$. Two triples (l_1, c_1, Y_1) and $l_2, c_2, Y_2)$ can combine iff $Y_1 = Y_2 = Y$. The resulting triple is $(\max\{l_1, l_2\}, c_1 + c_2 - \sum_{x \in Y} w(x), Y)$.

7.4. Complexity

The series-parallel decomposition of an SPDAG can be determined in $O(n)$ time [14]. By keeping each $F(G_i)$ as four separate lists of triples, one for each of the four possible values for the third coordinate of the triples, $F(G_1G_2)$ and $F(G_1//G_2)$ can be obtained in $O(|F(G_1)| \times |F(G_2)|)$ time from $F(G_1)$ and $F(G_2)$. Since $F(G_1)(F(G_2))$ contains only nondominated triples, it can contain at most four triples for each distinct value of the first coordinate and at most four for each distinct value of the second coordinate (these four must differ in their third coordinate). Hence, $|F(G_1)| \leq 4 \times \min\{\delta + 1, \sum_{u \in V(G_1)} w(u)\}$ and $|F(G_2)| \leq 4 \times \min\{\delta + 1, \sum_{u \in V(G_2)} w(u)\}$. So, we can obtain $F(G)$ for any SPDAG in time $O(n \times \min\{\delta^2, (sum_{u \in V(G)}\ w(u))^2\})$. For SPDAGs with unit delay or unit weight, this is $O(n^3)$.

8. DVUP(0, δ) FOR GENERAL SERIES PARALLEL DAGS

General series parallel dags (GSPDAGs) were introduced in [6, 8, 13]. A linear time algorithm to determine whether or not a given dag is a GSPDAG was developed in [14]. This paper also contains a linear time algorithm to obtain a series-parallel decomposition of a GSPDAG. The definitions and terminology used in this section are derived from [14].

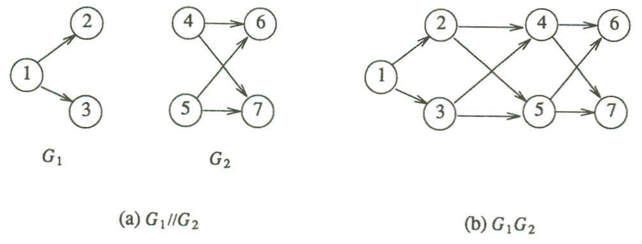

Fig. 14. Parallel and series combination of G_1 and G_2.

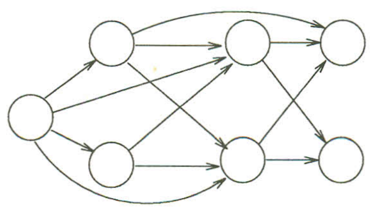

Fig. 15. An example GSPDAG.

A *transitive dag* is a dag $G = (V, E)$ such that $\langle i, j \rangle \in E$ whenever there is a path from i to j. The *transitive closure*, $G^+ = (V, E^+)$ of the dag $G = (V, E)$ is the transitive dag with $E \subseteq E^+$ and $|E^+|$ is minimum. An edge $\langle i, j \rangle$ of the dag G is *redundant* iff there is an i to j path in G that does not include $\langle i, j \rangle$. A dag is *minimal* iff it contains no redundant edges. The *transitive reduction*, $G^- = (V, E^-)$ of the dag $G = (V, E)$, is the minimal dag that has the same transitive closure as G and such that $E^- \subseteq E$.

The class of *minimal series-parallel dags* (MSPDAGs) is defined recursively as below:

1. If $|V| = 1$ and $|E| = 0$, then G is an MSPDAG.
2. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are MSPDAGs, then their parallel composition $G_1//G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ [Fig. 14(a)] is an MSPDAG.
3. If G_1 and G_2 are as above, then their series composition $G_1G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup E_3)$, where $E_3 = \{\langle i, j \rangle | i \in V_1, d^{out}(i) = 0, j \in V_2, d^{in}(j) = 0\}$, $d^{out}(i)$, and $d^{in}(j)$ are, respectively, the out- and in-degrees of vertex i [Fig. 14(b)], is an MSPDAG.

Finally, a GSPDAG is a dag whose transitive reduction is an MSPDAG. The dag of Figure 15 is a GSPDAG as its transitive reduction is the MSPDAG of Figure 14(b). It is interesting to note that every dag which is a tree is an MSPDAG. However, the only dag trees that are SPDAGs are chains.

The following lemma shows that the DVUP solutions for a GSPDAG G and for its transitive reduction MSPDAG G^- are the same.

Lemma 13. Let $G = (V, E)$ be a GSPDAG and let $G^- = (V, E^-)$ be its transitive reduction (hence, G^- is an MSPDAG). Let X be a subset of V. Then, $d(G|X) \leq \delta$ iff $d(G^-|X) \leq \delta$.

Proof. If $d(G|X) \leq \delta$, then $d(G^-|X) \leq \delta$ as $E^- \subseteq E$. If $d(G|X) > \delta$, then there is a path $P = v_1v_2 \cdots v_q$ in G such that its delay in $G|X$ is $\Delta = \sum_{1 \leq i \leq n, v_i \notin X} d(v_i)$ and $\Delta > \delta$. From the v_1 to v_q path P of G, we can construct a v_1 to v_q path Q in G^- by replacing each edge $\langle v_i, v_{i+1} \rangle \notin E^-$ by a path from v_i to v_{i+1} in G^-. Since G^- is the transitive reduction of G, such a path exists. When all edges $\langle v_i, v_{i+1}\rangle \notin E^-$ have been so replaced, we obtain the desired path Q. Clearly, the delay of Q is $\geq\Delta$. Hence, if $d(G|X) > \delta$, then $d(G^-|X) > \delta$. This completes the proof. ■

As a consequence of Lemma 13, the DVUP for GSPDAGs can be solved using the following steps:

STEP 1: Compute G^-, the transitive reduction of the GSPDAG G.

STEP 2: Let X be the minimum weight vertex subset such that $d(G^-|X) \leq \delta$.

STEP 3: Output X.

Since the transitive reduction of a GSPDAG G can be obtained in time linear in the number of vertices and edges in G [14], we need be concerned only with step 2. Our strategy for this is similar to that used in Section 3 for SPDAGs. However, since the source and sink vertices of G_1 and G_2 remain distinct following a series or parallel combination, we can deal with tuples (l, c) rather than with triples (l, c, Y). Corresponding to $D(l, Y, G)$ of SPDAGs, we define $D(l, G)$ for an MSPDAG G to be a minimum weight vertex set for which $d(G|X) \leq l$. We use the series-parallel decomposition of G and begin with the nondominated tuple sets $F(G)$ for each of the vertices in G. Then, using the series and parallel combinations that result in G, we obtain $F(G)$.

8.1. G Is a Single Vertex

If G is the single vertex v, then $F(G) = \{(0, w(v))\}$ when $d(v) > \delta$ and $\{(0, w(v)), (d(v), 0)\}$ when $d(v) \leq \delta$.

8.2. G Is of the Form G_1G_2

The following lemma is the analog of Lemma 11.

Lemma 14. If $(l, c) \in F(G_1G_2)$, then there is an $(l_1, c_1) \in F(G_1)$ and an $(l_2, c_2) \in F(G_2)$ such that

(a) $D(l_1, G_1) = D(l, G_1G_2) \cap V(G_1)$
(b) $D(l_2, G_2) = D(l, G_1G_2) \cap V(G_2)$
(c) $D(l, G_1G_2) = D(l_1, G_1) \cup D(l_2, G_2)$
(d) $c = \sum_{u\in D(l,Y,G_1G_2)} w(u)$, $c_1 = \sum_{u\in D(l_1,Y_1,G_1)} w(u)$, $c_2 = \sum_{u\in D(l_2,Y_2,G_2)} w(u)$.

Proof. Similar to that of Lemma 11. ■

For the case of MSPDAGs, all pairs $(l_1, c_1) \in F(G_1)$ and $(l_2, c_2) \in F(G_2)$ are compatible pairs for combination. The pair (l, c) that results from combining these two pairs is $(l_1 + l_2, c_1 + c_2)$.

8.3. $G = G_1//G_2$

The analog of Lemma 12 that applies to MSPDAGs is given below:

Lemma 15. If $(l, c) \in F(G_1//G_2)$, then there is an $(l_1, c_1) \in F(G_1)$ and an $(l_2, c_2) \in F(G_2)$ such that

(a) $D(l_1, G_1) = D(l, G_1G_2) \cap V(G_1)$
(b) $D(l_2, G_2) = D(l, G_1G_2) \cap V(G_2)$
(c) $D(l, G_1G_2) = D(l_1, G_1) \cup D(l_2, G_2)$
(d) $c = \sum_{u\in D(l,Y,G_1G_2)} w(u)$, $c_1 = \sum_{u\in D(l_1,Y_1,G_1)} w(u)$, $c_2 = \sum_{u\in D(l_2,Y_2,G_2)} w(u)$.

Proof. Similar to that of Lemma 11. ■

Once again, all pairs $(l_1, c_1) \in F(G_1)$ and $(l_2, c_2) \in F(G_2)$ are compatible pairs for combination. The result of combining these two pairs is the pair $(l, c) = (\max\{l_1, l_2\}, c_1 + c_2)$.

8.4. Complexity

While the algorithm for MSPDAGs is simpler than that for SPDAGs, its asymptotic complexity is the same.

9. CONCLUSIONS

In this paper, we have shown that the following problems are *NP*-hard:

1. LinkDelay(x, δ) for $\delta \neq 0$ or $x = 0$.
2. ShortestPath(x, δ) for $x = 0$ or $\delta > 0$.
3. Satellite(δ) for $\delta \geq 1$.
4. Longest Path (x, δ) for $x = 0$ and $\delta = 0$ and for $x \geq 0$ and $\delta > 0$.

In addition, we have obtained pseudopolynomial time algorithms for DVUP for trees, series-parallel dags, and general series-parallel dags. For trees with unit weights or unit delays, the complexity is $O(n^2)$ while it is $O(n^3)$ for series-parallel dags and general series-parallel dags with unit weights or unit delays. For the case of trees with unit weights and unit delays, we have developed a linear time algorithm.

REFERENCES

[1] P. K. Chan, Algorithms for library-specific sizing of combinational logic. *Proc.* 27*th DAC Conf.* (1990) 353–356.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability.* W. H. Freeman, San Francisco (1979).

[3] S. Ghanta, H. C. Yen, and H. C. Du, Timing analysis algorithms for large designs. University of Minnesota, Technical Report, 87-57 (1987).

[4] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Computer Science Press, New York (1978).

[5] M. Krishnamoorthy and N. Deo, Node deletion NP-complete problems. *SIAM J. Comput.* **8**(4) (1979) 619–625.

[6] E. L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann. Discr. Math.* **2** (1978) 75–90.

[7] P. McGeer, R. Brayton, R. Rudell, and A. Sangiovanni-Vincentelli, Extended stuck-fault testability for combinational networks. *Proceedings of the* 6th *MIT Conference on Advanced Research in VLSI.* MIT Press, Cambridge, MA (April 1990).

[8] C. L. Monma and J. B. Sidney, A general algorithm for optimal job sequencing with series-parallel constraints. Technical Report No. 347, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY (July 1977).

[9] D. Paik and S. Sahni, Upgrading circuit modules to improve performance. University of Florida, Technical report (1991).

[10] D. Paik, S. Reddy, and S. Sahni, Vertex splitting in dags and applications to partial scan designs and lossy circuits. University of Florida, Technical Report, 90-34 (1990).

[11] D. Paik, S. Reddy, and S. Sahni, Heuristics for the placement of flip-flops in partial scan designs and for the placement of signal boosters in lossy circuits. *Sixth International Conference On VLSI Design* (1993) 45–50.

[12] D. Paik, S. Reddy, and S. Sahni, Deleting vertices in dags to bound path lengths. *IEEE Trans. Comput.* **43**(9) (1994) 1091–1096.

[13] J. B. Sidney, The two machine flow line problem with series-parallel precedence relations. Working Paper 76-19, Faculty of Management Science, University of Ottawa (November 1976).

[14] J. Valders, R. E. Tarjan, and E. L. Lawler, The recognition of series parallel digraphs. *SIAM J. Comput.* **11** (1982) 298–313.

Received February 14, 1994
Accepted February 21, 1995