Performance Driven Graph Enhancement Problems*

Doowon Paik[a] and Sartaj Sahni[b]

[a]AT&T Bell Laboratories, Murray Hill, NJ
[b]Computer and Information Sciences Department, University of Florida, Gainesville, FL 32611

TECHNICAL REPORT: 92-036

Abstract

Graphs may be used to model systems in which performance issues are crucial. Cost effective performance enahancement of these systems can be accomplished by solving a graph enhancement problem on the associated graph. We define several graph enhancement problems. Some are shown to be NP-hard while others are polynomially solvable.

1 INTRODUCTION

When designing systems such as VLSI circuits or communication networks, one needs to make decisions that affect the performance of the resulting design. Often, the system is designed making one set of choices. The performance of the resulting design is determined. If this is found to be unsatisfactory, then one proceeds to change some of the design decisions so as to bring the system performance into the desired range. For example, we may design a circuit using certain circuit modules. Associated with each module is a delay. The circuit can be modeled as a directed acyclic graph (dag) with vertex weights. The vertices correspond to the circuit modules and the weights to the module delays. The sum of the vertex weights on any path gives the path length. The length of the longest path in the dag gives the circuit delay. If this delay exceeds the maximum allowable delay, then one can reduce the delay by choosing a different (and faster) implementation. However, choosing the faster implementation has a cost or weight associated with it. This results in a dag optimization problem: find a least weight vertex set whose upgrading results in a dag in which no path has length more than δ. In a simplified version of this problem, there is a factor x, $0 \le x < 1$ such that the upgraded module has a delay that is x times that of the original module. Let DVUP(x, δ) denote the dag vertex upgrade problem in which $d(i)$ is the delay of vertex i and $w(i)$ its weight.

In an alternate modeling of signal flow in electronic circuits by dags [CHAN90, GHAN87, MCGE90], vertices represent circuit modules and directed edges represent signal flow. In a simplistic model, each edge 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

+ This research was supported, in part, by the National Science Foundation under grant MIP 91-03379.

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 [PAIK91d]:

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. So, if edge $<v,w>$ has delay d before the upgrade, its delay is $x*d$ following the upgrade. If both v and w are upgraded, its delay becomes x^2*d.

As another example, consider a communication network. This 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. With respect to this undirected graph model, the following problems may be defined [PAIK91d]:

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, $0 \leq x < 1$. The problem is to upgrade the smallest number of vertices so that following the upgrades, no edge has delay $> \delta$.

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 $> \delta$.

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 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).

Each of the problems stated above is a simplified version of a more realistic problem. The more realistic problem has different costs associated with the upgrade of different vertices and the upgrade factor also varies from vertex to vertex.

Paik, Reddy, and Sahni [PAIK90, 93] model the optimal placement of scan registers in a partial scan design as well as the placement of signal boosters in lossy circuits as a vertex splitting problem in a dag. The input dag (which represents the circuit) has edge delays and the objective is to split the fewest number of vertices so that the resulting dag has no path of length > δ. When a vertex is split, it is replaced by two copies; one retains the incoming edges and the other the outgoing edges. The dag vertex splitting problem is denoted DVSP(δ).

The dag vertex deletion problem, DVDP(δ), is concerned with deleting the fewest number of vertices from an edge weighted dag so that the resulting dag has no path whose length exceeds δ. In [PAIK91a], Paik, Reddy, and Sahni used this problem to model the problem of upgrading circuit modules so as to control signal loss.

Krishnamoorthy and Deo [KRIS79] have shown that for many properties, the vertex deletion problem is NP-hard. These properties include: resulting graph has no edges, resulting graph is a clique, each component of the resulting graph is a tree, each component of the remaining graph is planar, etc. We shall not discuss any of these results here as none of the properties considered in [KRIS79] apply to graphs with vertex and/or edge weights.

In subsequent sections, we summarize the known results regarding the problems stated above. We shall make use of the following known NP-hard problems [GARE79].

1. Partition

Input: A set of n positive integers a_i, $1 \le i \le n$.

Output: "Yes" iff there is a subset, I, of $\{1, 2, ..., n\}$ such that

$$\sum_{i \in I} a_i = \sum_{i=1}^{n} a_i / 2$$

2. Vertex Cover

Input: An undirected graph $G = (V, E)$ and a positive integer $k \le |V|$

Output: "Yes" iff there is a subset $V' \subseteq V$ with $|V'| \le k$ such that for each edge $(u,v) \in E$ at least one of u and v belongs to V'.

3. Dominating Set

Input: An undirected graph $G = (V, E)$ and a positive integer $k \le |V|$

Output: "Yes" iff there is a subset $V' \subseteq V$ with $|V'| \le k$ such that for $u \in V - V'$ there is a $v \in V'$ for which $(u, v) \in E$.

4. Maximum Clique

Input: A connected undirected graph $G = (V, E)$ and a positive integer $k \le |V|$

Output: "Yes" iff there is a subset $V' \subseteq V$ with $|V'| \ge k$ such that two vertices in V' are joined by an edge in E.

5. Exact Cover By 3-Sets (X3C)

Input: Set X with $|X| = 3q$ and a collection $C = \{C_1, C_2, ..., C_m\}$ of three element subsets of X such that $\bigcup_{i=1}^{m} C_i = X$.

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'.

6. **3SAT Problem**

Input: A boolean function $F = C_1\ C_2 \cdots\ C_m$ in n variables $x_1, x_2, ..., x_n$. Each clause C_i is the disjunction of exactly three literals.

Output: "Yes" if there is a binary assignment for the n variables such that $F = 1$. "No" otherwise.

2 DVUP

Let $G = (V,E,w)$ be a weighted *directed acyclic graph* (wdag) with vertex set V, edge set E, and edge weighting funtion w. $w(i,j)$ is the weight of the edge $< i,j > \in E$. $w(i,j)$ is a positive integer for $< i,j > \in E$ and $w(i,j)$ is undefined if $< i,j > \notin E$. A *source vertex* is a vertex with zero in-degree while a *sink vertex* is a vertex with zero out-degree. The *delay*, $d(P)$, of the path P is the sum of the weights of the edges on that path. The delay, $d(G)$, of the graph G is the maximum path delay in the graph, i.e.,

$$d(G) = \max_{P \text{ in } G} \{ d(P) \}$$

Let $G \mid X$ be the wdag that results when the vertices of X are deleted from the wdag G. Note that the deletion of a vertex also requires the deletion of all incident edges.

It is easy to see that DVUP(0,δ) is NP-hard. In fact, the problem is NP-hard for dags that are chains. To prove this, we use the partition problem. Construct a chain of n vertices with $d(i) = w(i) = a_i$, $1 \le i \le n$. It is easy to see that there is an I such that $\sum_{i \in I} a_i = \sum_{i=1}^{n} a_i / 2$ iff the minimum cost X such that $d(G \mid X) \le \sum_{i=1}^{n} a_i / 2$ has cost $\sum_{i=1}^{n} a_i / 2$. Note that when $x = 0$, a vertex upgrade is equivalent to a vertex deletion. In the remainder of this section, we assume $x = 0$.

2.1 General Dags

2.1.1 Unit Delay Unit Weight DVUP

A dag $G = (V, E)$ is a *unit delay unit weight dag* iff $d(v) = w(v) = 1$ for every vertex $v \in V$. A subset X of V is *k-colorable* iff we can label the vertices of X using at most k labels and such that no two adjacent vertices have the same label. A *maximum k-coloring* of a dag G is a maximum subset $X \subseteq V$ that is k-colorable. A dag G is *transitive* iff for every $u, v, w \in G$ such that $<u, v> \in E$ and $<v, w> \in E$, the edge $<u, w>$ is also in E. $G^+ = (V, E^+)$ is the transitive closure of (V, E) iff $<u, v> \in E^+$ if there is a path (with at least one edge on it) in G from u to v. Note that if G is a dag then G^+ is a transitive dag.

The unit delay unit weight DVUP for any δ, $\delta \ge 1$ can be solved in $O(n^3 \log n)$ time by using the $O(n^3 \log n)$ maximum k-coloring algorithm of [GAVR87] for transitive dags. Before showing how this can be done, we establish a relationship between the number of vertices of a given set X that can be on

any path of G and colorability of X in G^+. Intuitively, X represents the set of vertices that are not upgraded. So, the number of X vertices on a path is the delay for that path.

Theorem 1: Let $G = (V, E)$ be a dag and let $G^+ = (V, E^+)$ be its transitive closure. Let X be a subset of the vertices in V. G has no path with $> \delta$ vertices of X iff X is δ colorable in G^+.

Proof: See [PAIK91b]. □

From the preceding theorem, it follows that if X is a maximum δ-coloring of G^+, $V–X$ is the smallest set such that $d(G \mid (V–X)) \leq \delta$. This implies the correctness of the following three step algorithm.

step 1: Compute G^+ from G
step 2: Compute X, a maximum δ-coloring of G^+
step 3: $B = V – X$ is the solution for the DVUP instance (G, δ)

The complexity of this is governed by that of step 2 which is $O(n^3 \log n)$. Note that when $\delta = 1$, a maximum δ-coloring is just a maximum independent set and such a set can be found in $O(ne)$ time for transitive closure graphs with n vertices and e edges [GAVR87]. So, the case $\delta \leq 1$ can be solved in $O(n^3)$ time as the graph G^+ computed in step 1 may have $O(n^2)$ edges even though G may not.

2.1.2 Nonunit Delay Unit Weight DVUP

A dag $G = (V, E)$ is a unit weight dag if $w(v) = 1$ for every $v \in V$. The case when $d(v)$ is also 1 for all v was considered in the previous section. So, in this section we are only concerned with the case of unit weight dags that have at least one vertex with delay > 1. In this section we show that the nonunit delay unit weight DVUP can be solved in $O(n^3)$ time when δ. The problem is NP-hard for $\delta \geq 2$ (see [PAIK91b] for a proof).

Let X be a minimum set of vertices such that $d(G \mid X) \leq 1$. Clearly, every $v \in V$ with $d(v) > 1$ must be in X. For each $v \in V$ with $d(v) > 1$, let $a_1^v, a_2^v, \ldots, a_q^v$ be the vertices such that $< a_i^v, v > \in E$ and let $b_1^v, b_2^v, \ldots, b_r^v$ be such that $< v, b_i^v > \in E$ and let G' be the dag that results when each such v (together with all edges incident to/from v) are deleted from G and all edges of the form $< a_i^v, b_j^v >$ are added. To get G', this transformation is applied serially to all v with $d(v) > 1$.

Let $B = \{v \mid d(v) > 1 \text{ and } v \in V\}$. Let $G' = (V', E')$ and let $C \subseteq V'$ be a minimum vertex set such that $d(G' \mid C) \leq 1$. It is easy to see that $A = B \cup C$ is a minimum vertex set such that $d(G \mid X) \leq 1$. C can be obtain in $O(n^3)$ time using the unit delay unit weight algorithm (note that G' is a unit delay unit weight dag), B can be obtained in $O(n)$ time, and G' can be constructed in $O(n^3)$ time. So, the overall complexity of our algorithm to compute X is $O(n^3)$.

Theorem 2: Non unit delay unit weight DVUP is NP-hard for every δ, $\delta \geq 2$.

Proof: See [PAIK91b]. Since the construction of [PAIK91b] generates a multistage graph, DVUP is NP-hard even when the dags are restricted to be multistage graphs. □

2.2 Trees

2.2.1 Trees With Unit Weight And Unit Delay

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 $d(T \mid X) \leq \delta$ 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 [HORO90]. The correctness of the procedure outlined above is established in Theorem 3. Note that when all vertices have unit weight, the weight of a set, Y, of vertices is simply its cardinality $|Y|$.

Theorem 3: For any tree T let $h(v)$ be the height of vertex v. The set

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

is a minimum cardinality vertex set such that $d(T \mid X) \leq \delta$.

Proof: The fact that $d(T \mid X) \leq \delta$ is easily seen. The minimality of X is by induction on the number, n, of vertices in T. For the induction base, we see that when $n = 0$, $|X| = 0$ and the theorem is true. Assume the theorem is true for all trees with $\leq m$ vertices where m is an arbitrary natural number. We shall see that the theorem is true when $n = m + 1$. Consider any tree with $n = m + 1$ vertices. Let X be the set of all vertices with height $> \delta$. If $|X| = 0$, then X is clearly a minimal set with $d(T \mid X) \leq \delta$. Assume $|X| > 0$. In this case the root, r, of T has $h(r) > \delta$ and so is in X. First, we show that there is a minimal vertex set W that contains r and for which $d(T \mid W) \leq \delta$. Let Z be a minimal vertex set for which $d(T \mid Z) \leq \delta$. If $r \notin Z$, then let $r, u_1, u_2, \ldots, u_{h(r)-1}$ be a longest root to leaf path in T. Since $h(r) > \delta$, at least one of the u_i's is in Z. Let u_j be any one of the u_i's in Z. Let $W = Z + \{r\} - \{u_j\}$. Clearly, $|W| = |Z|$. Since all root to leaf paths that include u_j also include r, the length of these paths in $T \mid W$ is the same as in $T \mid Z$. The length of the remaining paths in $T \mid W$ is no more than in $T \mid Z$. So, W is a minimal cardinality vertex set such that $d(T \mid W) \leq \delta$ and furthermore W contains the root r.

Let $A(v)$, $A \in \{X, W\}$, denote the subset of A that consists only of vertices in the subtree, $T(v)$, rooted at v. Since $d(T(v) \mid X(v)) \leq \delta$, $d(T(v) \mid W(v)) \leq \delta$, and $|T(v)| \leq m$ for each v that is a child of r, it follows from the induction hypothesis that $|X(v)| = |W(v)|$ for each v that is a child of r. Hence, $|X| = 1 + \sum_{v \text{ is a child of } r} |X(v)| = |W|$. □

2.2.2 General Trees

Since a chain is a special case of a tree and since DVUP for chains with arbitrary weights and delays has been shown to be NP-hard, we do not expect to find a polynomial time algorithm for general trees. In this section we develop a pseudo polynomial time algorithm (i.e., one whose complexity is polynomial in the number of vertices and the actual values of the vertex delays and weights). We modify the definition of height used in the preceding section to account for the vertex delays. We use H to denote this modified height.

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

For each vertex v, let $L(v)$ be a set of pairs (l, c) such that there is a subset $X \subseteq T(v)$ such that $d(T(v) \mid X) = l \le \delta$ and $\sum_{u \in X} w(u) = c$. Let (l_1, c_1) and (l_2, c_2) be two different pairs such that $l_1 \le l_2$ and $c_1 \le c_2$. In this case, pair (l_1, c_1) *dominates* (l_2, c_2). Let $S(v)$ be the subset of $L(v)$ that results from the deletion of all dominated pairs. Let $S(r)$ be this set of dominating pairs for the root r of T. Let $(l', c') \in S(r)$ be the pair with least cost c'. It is easy to see that the least weight vertex set W such that $d(T \mid W) \le \delta$ has weight c'. We shall describe how to compute $S(r)$. Using the backtrace strategy of [HORO78, cf. chapter on dynamic programming] we can compute the W that results in $d(T \mid W) \le \delta$ and $\sum_{u \in W} w(u) = c'$ in less time than needed to compute $S(r)$ (however, $S(r)$ and some of the other S's computed while computing $S(r)$ are needed for this).

For a leaf vertex v, $S(v)$ is $\{(0, w(v))\}$ when $d(v) > \delta$ and $\{(0, w(v)), (d(v), 0)\}$ otherwise. For a non leaf 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 $(l_i, c_i) \in S(u_i)$, $1 \le i \le k_v$. Let $V(v)$ and $Y(v)$ be as below:

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

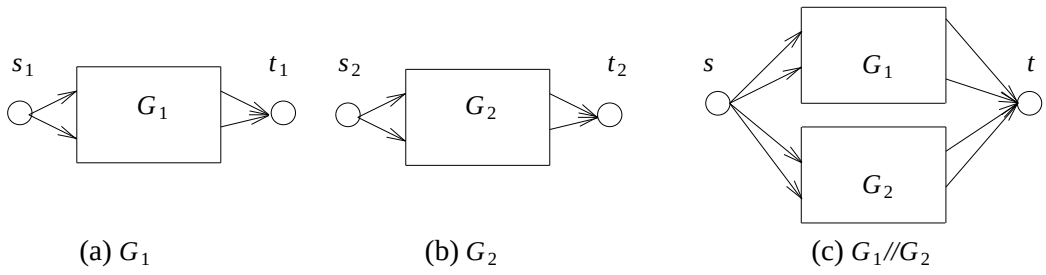
Now, $S(v)$ is the set of nondominated pairs in $V(v) \cup Y(v)$. Since $S(v)$ contains only nondominated pairs, all pairs in $S(v)$ have different l and c values. So, $|S(v)| \le \min\{ \delta, \omega \}$ where $\omega = \sum_{u=1}^{n} w(u)$. Using the technique of [HORO78], $S(v)$ can be computed from the $S(u)$'s of its children in time $O(\min\{\delta, \omega\} * k_v)$. To compute $S(r)$ we need to compute $S(v)$ for all vertices v. The time needed for this is $O(\min\{\delta, \omega\} * \sum k_v) = O(\min\{\delta, \omega\} * n)$

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

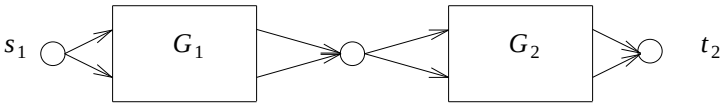
2.3 Series-Parallel Dags

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 (Figure 1(c)). $G_1 // G_2$ is an SPDAG. We restrict G_1 and G_2 so that at most one of the edges $< s_1, t_1 >$, $< s_2, t_2 >$ 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 (Figure 1(d)). $G_1 G_2$ is an SPDAG.



(a) G_1 (b) G_2 (c) $G_1 // G_2$



(d) $G_1 G_2$

Figure 1: Series-parallel digraph.

The strategy we employ for SPDAGs is a generalization of that used for trees with general delays and weights. Let s and t, respectively, be the source and sink vertices of the SPDAG G. Let $D(l,Y,G)$ be a minimum weight vertex set that contains the vertices in Y, $Y \subseteq \{s, t\}$, and such that $d(G \mid D(l,Y,G)) \leq l$ and let $f(G)$ be as below:

$$f(G) = \{ (l,c,Y) \mid 0 \leq l \leq \delta, \quad c = \sum_{u \in D(l,Y,G)} w(u) \}.$$

Let (l_1, c_1, Y_1) and (l_2, c_2, Y_2) be two different triples in $f(G)$. (l_1, c_1, Y_1) *dominates* (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 $f(G)$. If (l', c', Y') is the least weight triple (i.e., the one with least c) in $F(G)$ then the least weight

W such that $d(G \mid W) \leq \delta$ has weight c'. We shall show how to compute $F(G)$ and hence (l', c', Y'). The actual W may be obtained using a backtrace step as described in [HORO78].

2.3.1 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.

2.3.2 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 1: 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: See [PAIK91c]. □

Lemma 1 suggests the following approach to obtain $F(G_1G_2)$ from $F(G_1)$ and $F(G_2)$:

Step1: Construct a set Z of triples such that $F(G_1G_2) \subseteq Z$. This is obtained by combining together all pairs of triples $(l_1, c_1, Y_1) \in F(G_1)$ and $(l_2, c_2, Y_2) \in F(G_2)$.

Step2: Eliminate from Z all triples that are dominated by at least one other triple of Z.

The triples (l_1, c_1, Y_1) and (l_2, c_2, Y_2) are compatible iff ($t_1 \in Y_1$ and $s_2 \in Y_2$) or ($t_1 \notin Y_1$ and $s_2 \notin Y_2$). Only compatible triples may be combined. Assume that we are dealing with two compatible triples. We first obtain the triple (l, c, Y) as below:

if $t_1 \in Y_1$
then $(l, c, Y) := (l_1+l_2, c_1+c_2-w(t_1), Y_1 \cup Y_2 - \{t_1\})$
else $(l, c, Y) := (l_1+l_2-d(t_1), c_1+c_2, Y_1 \cup Y_2 - \{t_1\})$

Next, (l, c, Y) is added to Z provided $l \leq \delta$.

2.3.2.1 $G = G_1 // G_2$

When $G = G_1 // G_2$ we use Lemma 2 which is the analogue of Lemma 1.

Lemma 2: 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) $D(l_1, Y_1, G_1) = D(l, Y, G_1 G_2) \cap V(G_1)$

(b) $D(l_2, Y_2, G_2) = D(l, Y, G_1 G_2) \cap V(G_2)$

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

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

Proof: Similar to that of Lemma 1. □

To obtain $F(G_1 // G_2)$ from $F(G_1)$ and $F(G_2)$ we use the two step approach used to compute $F(G_1 G_2)$. For step 1, we compute the triple (l, c, Y) obtained by combining $(l_1, c_1, Y_1) \in F(G_1)$ and $(l_2, c_2, Y_2) \in F(G_2)$. The triples are compatible iff $Y_1 = Y_2$. Again, only compatible triples may be combined. For compatible triples, (l, c, Y) is obtained as below:

$l := \max \{l_1, l_2\}$

$l := c_1 + c_2 - \sum_{u \in Y_1} w(u)$

$Y := Y_1$

Next, (l, c, Y) is added to Z.

2.3.3 Complexity

The series-parallel decomposition of an SPDAG can be determined in $O(n)$ time [VALD79]. 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, $F(G_1 G_2)$ and $F(G_1 // G_2)$ can be obtained in $O(|F(G_1)| * |F(G_2)|)$ time from $F(G_1)$ and $F(G_2)$. Since $F(G_1)$ ($F(G_2)$) contains only non dominated 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, $|F(G_1)| \leq 4*\min \{ \delta + 1, \sum_{u \in V(G_1)} w(u) \}$ and $|F(G_2)| \leq 4*\min \{ \delta + 1, \sum_{u \in V(G_2)} w(u) \}$. So, we can obtain $F(G)$ for any SPDAG in time $O(n*\min \{\delta^2, (\sum_{u \in V(G)} w(u))^2\})$. For SPDAGs with unit delay or unit weight, the complexity is $O(n^2)$. To see this, note that for unit weight or unit delay SPDAGs, $|F(G_1)| \leq 4|V(G_1)|$ and $|F(G_2)| \leq 4|V(G_2)|$. So, $t(G) \leq t(G_1) + t(G_2) + 16|V(G_1)|*|V(G_2)|$.

2.3.4 Extension To General Series Parallel Dags

The algorithm for series parallel dags may be extended to obtain an algorithm of the same asymptotic complexity for general series parallel dags (GSPDAG). These were introduced in [LAWL78, MONM77, SIDN76]. The extension may be found in [PAIK91c].

3 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 4: [PAIK91d] LinkDelay(x,δ) is NP-hard whenever $\delta \neq 0$ or $x = 0$.

Proof: Let $G = (V, E)$ be an instance of the vertex cover 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 $\leq k$. □

[PADH92] considers the link delay problem for trees and series-parallel graphs. For both cases, she develops a linear time algorithm. The case for trees is easily solved using a tree traversal algorithm. First, note that when $\delta = 0$ and $x > 0$, the problem has a solution iff all edges have zero delay. In this case, no vertex is to be upgraded. A simple examination of the edge delays suffices to solve this case. When $\delta \geq 0$ and $x = 0$, we begin by removing all edges with delay less than δ. This leaves behind a forest of trees. Trees with a single node are removed from consideration. From each of the remaining trees, the parents of all leaves are marked for upgrading and edges incident to these parents deleted (as their delay is now zero). The result is a new forest from which single node trees are removed and the parents of leaves upgraded. This process continues until the forest becomes empty. When, $\delta > 0$ and $x > 0$, we first verify that there is no edge with delay greater than δ/x^2. This is a necessary and sufficient condition for the existence of a solution. Next, for each edge (u, v) with delay d such that $x*d > \delta \geq x^2*d$, both u and v are upgraded. Following this, we remove all edges with delay $\leq \delta$ from the tree and upgrade vertices in the resulting forest using the forest upgrading scheme just described for the case $\delta \geq 0$ and $x = 0$. The correctness of the algorithm is easily established and using appropriate data structures, it can be implemented to have complexity $O(n)$ where n is the number of vertices in the tree.

For the case of series-parallel graphs, [PADH92] proposes a dynamic programming algorithm which uses the series-parallel decomposition of the graph. For each (series-parallel) graph in the decomposition, she keeps track of the best solution that (a) necessarily upgrades the source vertex but not the sink, (b) necessarily upgrades the sink vertex but not the source, (c) necessarily upgrades both the source and sink, and (d) upgrades neither the source nor the sink. Since only four solutions are recorded in each stage of the decomposition, the resulting dynamic programming algorithm has complexity $O(n)$ where n is the number of vertices in the input series-parallel graph.

4 ShortestPath(x,δ)

We note that while at first glance ShortestPath(0,0) may appear to be identical to either the vertex cover or the dominating set problem, this is not so.

Theorem 5 : [PAIK91d] ShortestPath(x,δ) is NP-hard whenever $x = 0$ or $\delta > 0$.

Proof: We shall prove this here only for the case $x = \delta = 0$. Let X, q, C, and m be as in the definition of X3C. Construct an instance $G = (V, E)$ of ShortestPath(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 $C_1, C_2, ..., C_m, Z_1, Z_2, ..., Z_{q+2}$. These are the level 2 nodes of the graph. Child C_i represents set C_i, $1 \le i \le m$ 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, ..., $3q$. Node i represents element i of X, $1 \le i \le 3q$.

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, $1 \le i \le m$.

We shall show that the input X3C instance has answer "yes" iff the ShortestPath(0,0) instance G has an upgrade sets of size $\le q + 1$. First, suppose that the answer to the X3C instance is "yes". Then there is a $C' \subseteq C$ such that C' is an exact cover of X. Since $|X| = 3q$ and $|C_i| = 3$, $1 \le i \le m$, $|C'| = q$. 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 $\le q + 1$. 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 three 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 $\le q$ vertices in S. The only way this can be accomplished by an upgrade of $\le q$ vertices is if these remaining vertices are a subset of $\{C_1, C_2, ..., C_m\}$ 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 $\le q + 1$. □

5 Satellite(δ)

Satellite(0) is trivially solved. First, zero length edges are eliminated by combining their endpoints 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 3 : 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 non-satellite vertices). CommTime(G) ≤ 1 iff the following are true:

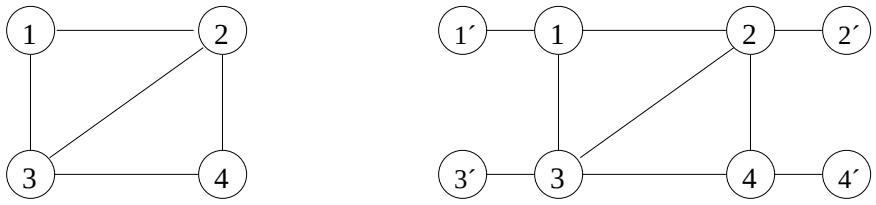
(a) The vertices of N define a clique of G.

(b) Each vertex of N is adjacent to a vertex of S (i.e., S is a dominating set of G) unless $S = \emptyset$.

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 $S \neq \emptyset$. 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. □

Theorem 6: [PAIK91d] Satellite(δ) is NP-hard for $\delta \geq 1$.

Proof: We shall only present the proof for the case $\delta = 1$. The complete proof can be found in [PAIK91d]. The proof uses the max clique problem. 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 (Figure 2). The number of vertices in G' is $2n$. All edges of G' have unit cost. 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 3 on non-satellite 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 3 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 3), 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. □



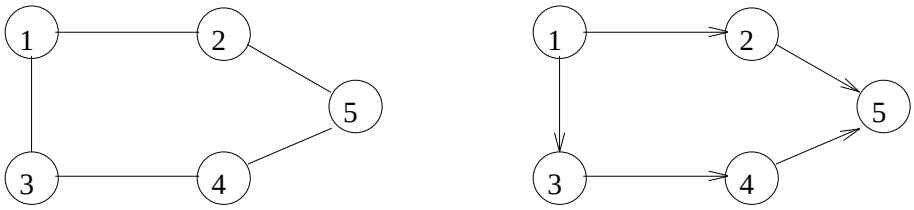
(a) G (b) G'

Figure 2: Construction of Theorem 6.

6 LongestPath(x,δ)

Lemma 4: LongestPath (0,0) is NP-hard.

Proof: Let G be a connected undirected graph. We shall construct an instance G' of LongestPath(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 higher index vertex. I.e., if (i, j), $i < j$, is an edge of G, then $<i, j>$ is a directed edge of G' (Figure 3). 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'. □



(a) G' (b) G

Figure 3: Construction for Lemma 4.

To show that LongestPath(x,δ) is NP-hard for $x = 0$ and $\delta > 0$, we use the subgraph $J_{n,q}$ (Figure 4) which is comprised of n directed chains of size q that share a common vertex r. This set of n chains is connected to a two vertex directed chain $<r, s>$. 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 5: 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 (Figure 5) 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 7: [PAIK91d]

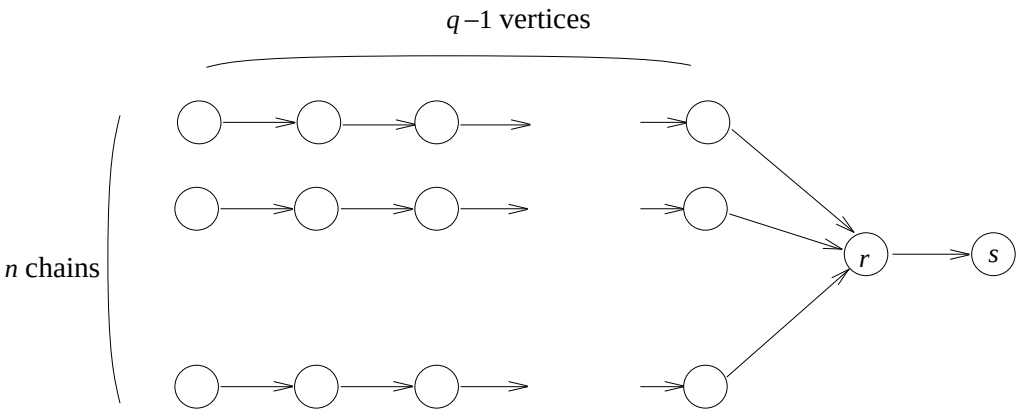


Figure 4: $J_{n,\,q}$.

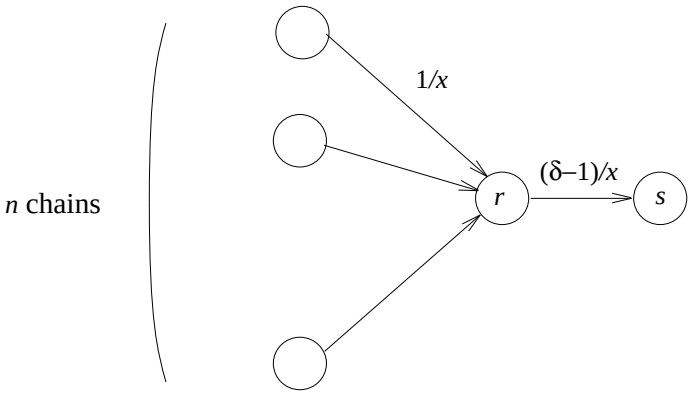


Figure 5: Q_n.

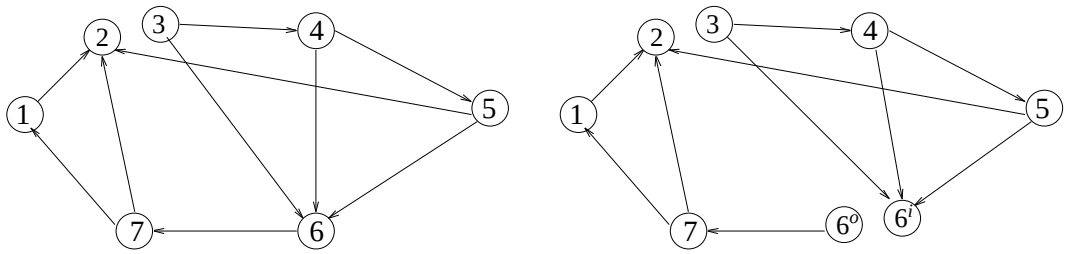
(a) Longest Path (x,δ) is NP-hard when $x = 0$ and $\delta = 0$ and also when $x \geq 0$ and $\delta > 0$.

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

Proof: (a) has been proved in Lemmas 4 and 5. 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. □

7 DVSP

Let G/X be the wdag that results when each vertex v in X is split into two v^i and v^o such that all edges $< v,j > \in E$ are replaced by edges of the form $< v^o,j >$ and all edges $< i,v > \in E$ are replaced by edges of the form $< i,v^i >$. I.e., outbound edges of v now leave vertex v^o while the inbound edges of v now enter vertex v^i. Figure 6(b) shows the result, G/X, of splitting the vertex 6 of the dag of Figure 6(a). The *dag vertex splitting problem* (DVSP) is to find a least cardinality vertex set X such that $d(G/X) \leq \delta$, where δ is a prespecified delay. For the dag of Figure 6(a) and $\delta = 3$, $X = \{6\}$ is a solution to the DVSP problem.



(a) A dag (b) Vertex 6 split

Figure 6: Example of vertex splitting

7.1 Complexity Results

If $w(i,j) = 1$ for every edge in the wdag, then the edge weighting function w is said to be a *unit weighting function* and we say that G has unit weights. When $\delta = 1$, the unit weight DVSP can be solved in linear time as every vertex that is not a source or sink has to be split. However, for every $\delta \geq 2$, the problem is NP-hard [PAIK90]. To show this, for each instance F of 3SAT, we construct an instance G_F of the unit weight DVSP such that from the size of the solution to G_F we can determine, in polynomial time, the answer to the 3SAT problem for F. This construction employs two unit weight dag subassemblies: variable subassembly and clause subassembly.

Variable Subassembly

Figure 7(a) shows a chain with $\delta - 1$ vertices. This chain is represented by the schematic of Figure 7(b). The variable subassembly, $VS(i)$, for variable x_i is given in Figure 7(c). This is obtained by combining together three copies of the chain $H_{\delta-1}$ with another chain that has three vertices. Thus, the total number of vertices in the variable subassembly $VS(i)$ is 3δ. Note that $d(VS(i)) = \delta + 1$. Also, note that if $d(VS(i)/X) \leq \delta$, then $|X| \geq 1$. The only X for which $|X| = 1$ and $d(VS(i)/X) \leq \delta$ are $X = \{ x_i \}$ and $X = \{ \bar{x}_i \}$. Figure 7(d) shows the schematic for $VS(i)$.

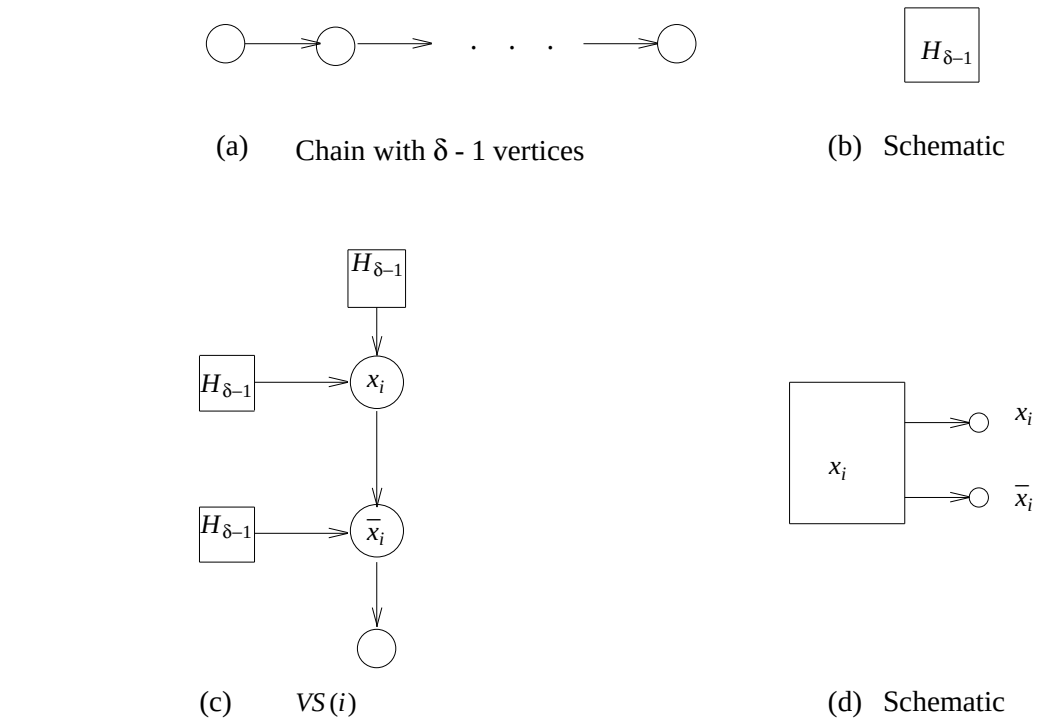


Figure 7: Variable subassembly for DVSP.

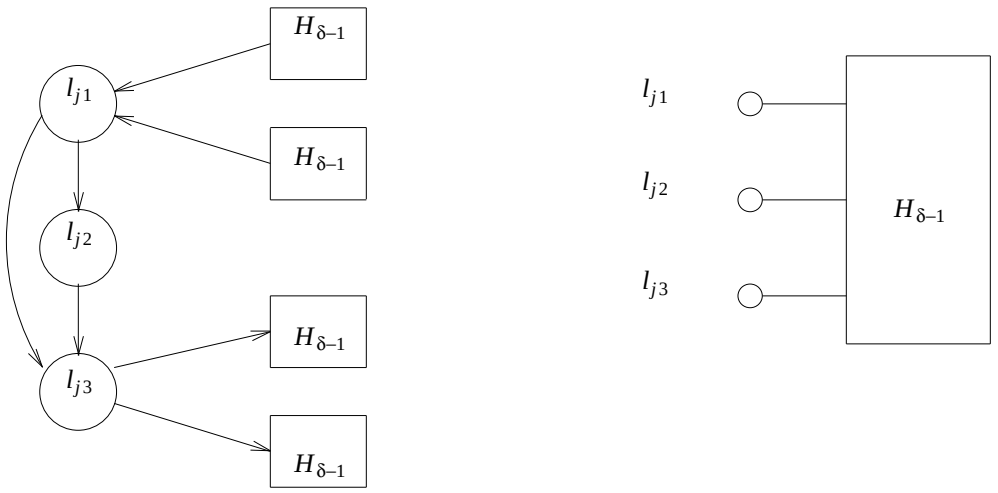
Clause Subassembly

The clause subassembly $CS(j)$ is obtained by connecting together four $\delta - 1$ vertex chains with another three vertex subgraph as shown in Figure 8(a). The schematic for $CS(j)$ is given in Figure 8(b). The number of vertices in $CS(j)$ is $4\delta - 1$ and $d(CS(j)) = 2\delta$. One may easily verify that if $|X| = 1$, then $d(CS(j)/X) > \delta$. So, if $d(CS(j)/X) \leq \delta$,then $|X| > 1$. Since $\delta \geq 2$, the only X with $|X| = 2$ for which $d(CS(j)/X) \leq \delta$ are such that $X \subseteq \{l_{j1}, l_{j2}, l_{j3}\}$. Furthermore, every $X \subseteq \{l_{j1}, l_{j2}, l_{j3}\}$ with $|X| = 2$ results in $d(CS(j)/X) \leq \delta$.

To construct G_F from F, we use n $VS(i)$'s, one for each variable x_i in F and m $CS(j)$'s, one for each clause C_j in F. There is a directed edge from vertex x_i ($\overline{x}_i$) of $VS(i)$ to vertex l_{jk} of $CS(j)$ iff x_i ($\overline{x}_i$) is the k'th literal of C_j (we assume the three literals in C_j are ordered). For the case $F = (x_1 + \overline{x}_2 + \overline{x}_4)$ $(\overline{x}_1 + \overline{x}_3 + \overline{x}_4)$ $(x_1 + x_2 + x_3)$, the G_F of Figure 9 is obtained.

Since the total number of vertices in G_F is $3\delta n + (4\delta - 1)m$, the construction of G_F can be done in polynomial time for any fixed δ.

Theorem 8: Let F be an instance of 3SAT and let G_F be the instance of unit weight DVSP obtained using the above construction. For $\delta \geq 2$, F is satisfiable iff there is a vertex set X such that $d(G_F/X) \leq \delta$ and $|X| = n + 2m$.



(a) $CS(j)$ (b) Schematic

Figure 8: Clause assembly for DVSP.

Proof: If F is satisfiable then there is a binary assignment to the x_i's such that F has value 1. Let b_1,b_2, ... b_n be this ssignment. Construct a vertex set X in the following way:

1. x_i is in X if $b_i = 1$. If $b_i = 0$, then $\bar{x}_i$ is in X.
2. From each $CS(j)$ add exactly two of the vertices l_{j1}, l_{j2}, l_{j3} to X. These are chosen such that the literal corresponding to the vertex not chosen has value 1. Each clause has at least one literal with value 1.

We readily see that $|X| = n + 2m$ and that $d(G_F/X) \leq \delta$.

Next, suppose that there is an X such that $|X| = n + 2m$ and $d(G_F/X) \leq \delta$. From the construction of the variable and clause assemblies and from the fact that $|X| = n + 2m$, it follows that X must contain exactly one vertex from each of the sets $\{x_i, \bar{x}_i\}$, $1 \leq i \leq n$ and exactly 2 from each of the sets $\{l_{j1}, l_{j2}, l_{j3}\}$, $1 \leq j \leq m$. Hence there is no i such that both $x_i \in X$ and $\bar{x}_i \in X$ and there is no j for which $l_{j1} \in X$ and $l_{j2} \in X$ and $l_{j3} \in X$. Consider the Boolean assignment $b_i = 1$ iff $x_i \in X$. Suppose that $l_{jk} \notin X$ and $l_{jk} = x_i$ ($\bar{x}_i$). Since $d(G_F/X) \leq \delta$, vertex x_i ($\bar{x}_i$) must be split as otherwise there is a source to sink path with delay greater than δ. So, x_i ($\bar{x}_i$) $\in X$ and $b_i = 1$ (0). As a result, the k'th literal of clause C_j is true. Hence, b_1, ... b_n results in each clause having at least one true literal and F has value 1. □

Theorem 9: DVSP is NP-hard for unit weight multistage graphs when $\delta \geq 4$.

Proof: See [PAIK90]. □

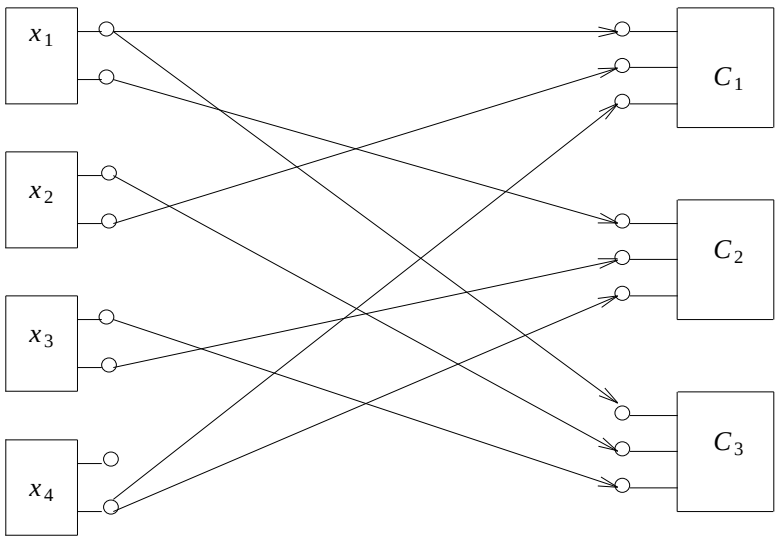


Figure 9: G_F for $F = (x_1 + \bar{x}_2 + \bar{x}_4)\ (\bar{x}_1 + \bar{x}_3 + \bar{x}_4)\ (x_1 + x_2 + x_3)$.

7.2 Polynomially Solvable Cases

When the wdag G is a rooted tree the DVSP problem can be solved in linear time by performing a postorder [HORO90] traversal of the tree. During this traversal we compute, for each node x, the maximum delay, $D(x)$, from x to any other node in its subtree. If x has a parent z and $D(x) + w(z,x)$ exceeds δ, then the node x is split and $D(z)$ is set to 0. Note that $D(x)$ satisfies:

$$D(x) = \max_{y \text{ is a child of } x} \{ D(y) + w(x,y) \}$$

Another polynomially sovlable case is when the dag is a series-parallel graph. For such graphs dynamic programming can be used to obtain a quadratic time algorithm [PAIK91a]. For general dags, a backtracking algorithm has been formulated in [PAIK90] and heuristics have been developed and evaluated in [PAIK90, 93].

8 DVDP

Let G be a wdag as in the previous section and let X be a subset of the vertices of G. Let $G-X$ be the wdag obtained when the vertices in X are deleted from the wdag G. This vertex set deletion is also accompanied by the deletion of all edges in G that are incident to a deleted vertex. The *dag vertex deletion problem* (DVDP) is to find a least cardinality vertex set X such that $d(G-X) \le \delta$, where δ is a prespecified graph delay.

Lemma 6: Let $G = (V,E,w)$ be a weighted wdag and let δ be a prespecified delay value. Let MaxEdgeDelay $= \max_{<i,j> \in E} \{ w(i,j) \}$.

(a) The DVDP has a solution iff $\delta \geq 0$.

(b) The DVSP has a solution iff $\delta \geq$ MaxEdgeDelay.

(c) For every $\delta \geq$ MaxEdgeDelay, the size of the DVDP solution is less than or equal to that of the DVSP solution.

Proof:

(a) Since $d(G-V) = 0$, there must be a least cardinality set X such that $d(G-X) \leq \delta$.

(b) Vertex splitting does not eliminate any edges. So, there is no X such that $d(G/X) <$ MaxEdgeDelay. Further, $d(G/V) =$ MaxEdgeDelay. So, for every $\delta \geq$ MaxEdgeDelay, there is a least cardinality set X such that $d(G/X) \leq \delta$.

(c) Let X be a solution to the DVSP. Since $d(G/X) \leq \delta$, $d(G-X) \leq \delta$. Hence the cardinality of the DVDP solution is $\leq |X|$. □

Let $|DVSP|$ ($|DVDP|$) be the size of solution to the DVSP (DVDP).

Lemma 7: For every δ, $\delta \geq 0$, there is a wdag $G = (V,E,w)$ with MaxEdgeDelay $\leq \delta$ such that $|DVSP| / |DVDP|$ = number of nodes that are neither source nor sink.

Proof: Consider the wdag of Figure 10. $d(G - \{v\}) = \delta$. However, since every edge has weight δ, it is necessary to split every vertex that is not a source or sink to get the delay down to δ. □

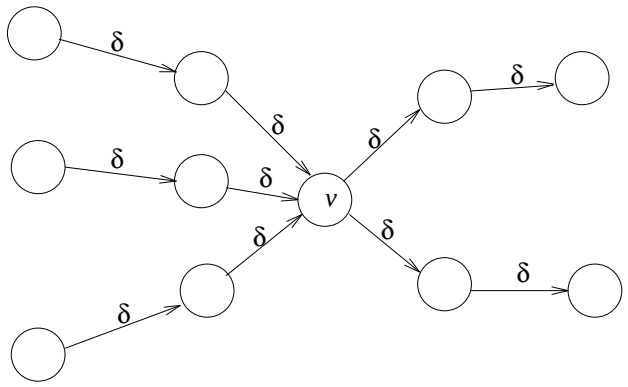


Figure 10: Construction for Lemma 7.

Corollary 1: For every $\delta \geq$ MaxEdgeDelay and every wdag G such that $d(G) > \delta$, $1 \leq |DVSP| / |DVDP| \leq$ number of nodes that are neither source nor sink.

Proof: The lower bound follows from Lemma 6 part (c) and the upper bound follows from the observation that $|DVSP| \leq$ number of nodes that are neither source nor sink and $|DVDP| \geq 1$. Note that the source and sink vertices of a wdag never need to be split. □

8.1 Complexity Results

Paik, Reddy, and Sahni [PAIK91a] have shown that the DVDP problem is NP-hard for unit weight dags with $\delta \geq 0$ as well as for unit weight multistage graphs with $\delta \geq 2$. We shall present only the proof for the case of unit weight dags and $\delta = 0$. The interested reader is referred to [PAIK91a] for the remaining proofs.

Theorem 10: Unit weight DVDP is NP-hard for $\delta = 0$.

Proof: Let G be an instance of unit weight DVDP and let X be such that $d(G - X) = 0$. So, X must contain at least one of the two end-points of each edge of G. Hence, X is a vertex cover of the undirected graph obtained from G by removing directions from the edges. Actually, every vertex cover problem can be transformed into an equivalent DVDP with $\delta = 0$. Let U be an arbitrary undirected graph. Replace each undirected edge (u,v) of U by the directed edge $< \min\{u,v\},\max\{u,v\}>$ to get the directed graph V. V is a wdag as one cannot form a cycle solely from edges of the form $< i,j>$ where $i < j$. Furthermore the DVDP instance V with $\delta = 0$ has a solution of size $\leq k$ iff the corresponding vertex cover instance U does. Hence, unit weight DVDP with $\delta = 0$ is NP-hard. □

8.2 Polynomially Solvable Cases

As in the case of the DVSP problem, the DVDP problem can be solved in linear time when the wdag is a tree and in quadratic time when the wdag is a series-parallel graph. The algorithms are similar to those for the corresponding DVSP cases and can be found in [PAIK91a].

9 REFERENCES

[CHAN90] P. K. Chan, "Algorithms For Library-Specific Sizing Of Combinational Logic", *Proc. 27th DAC Conf.*, 1990 pp. 353-356.

[GARE79] M. R. Garey, and D. S. Johnson, "Computers and Intractability", W. H. Freeman and Company, San Francisco, 1979.

[GAVR87] F. Gavril, "Algorithms For Maximum k-colorings And k-coverings Of Transitive Graphs", *Networks*, Vol. 17, pp. 465-470, 1987.

[GHAN87] S. Ghanta, H. C. Yen, and H. C. Du, "Timing Analysis Algorithms For Large Designs", University of Minnesota, Technical Report, 87-57,1987.

[HORO78] E. Horowitz, and S. Sahni, "Fundamentals of Computer Algorithms", Computer Science Press, Maryland, 1978.

[KRIS79] M. Krishnamoorthy and N. Deo, "Node deletion NP-complete problems", SIAM Jr on Computing, Vol 8, No 4, 1979, pp 619-625.

[LAWL78] E. L. Lawler, "Sequencing Jobs To Minimize Total Weighted Completion Time subject to precedence constraints", *Annals of Discrete Math. 2*, 1978, 75-90.

[LEE90] D. H. Lee and S. M. Reddy, "On Determining Scan Flip-flops In Partial-scan Designs", *Proc. of International Conference on Computer Aided Design*, November 1990.

[MCGE90] P. McGeer, R. Brayton, R. Rudell, and A. Sangiovanni-Vincentelli, "Extended Stuck-fault Testability For Combinational Networks", *Proc. of the 6th MIT Conference on Advanced Research in VLSI*, MIT Press, April 1990.

[MONM77] 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, N.Y., July 1977.

[PADH92] V. Padhye, "Upgrading vertices in trees and series-parallel graphs to bound link delays," University of Florida, Sept. 1992.

[PAIK90] 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.

[PAIK91a] D. Paik, S. Reddy, and S. Sahni, "Deleting Verticies To Bound Path Lengths", University of Florida, Technical Report, 91-4, 1991.

[PAIK91b] D. Paik, and S. Sahni, "Upgrading Circuit Modules To Improve Performance", University of Florida, Technical Report, 1991.

[PAIK91c] D. Paik, and S. Sahni, "Upgrading Vertices In Trees, Series-Parallel Digraphs And General Series Parallel Digraphs", University of Florida, Technical Report, 1991.

[PAIK91d] D. Paik, and S. Sahni, "NP-hard Network Upgrading Problems", University of Florida, Technical Report, 1991.

[PAIK93] D. Paik, S. Reddy, and S. Sahni, "Heuristics for the placement of flip-flops in partial scan designs and the placement of signal boosters in lossy circuits", *Proc. VLSI Design '93*, IEEE, 1993.

[SIDN76] 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.

[VALD79] J. Valders, R. E. Tarjan, and E. L. Lawler, "The recognition of Series Parallel digraphs", *SIAM J. Comput.*, 11 (1982), pp. 298-313.