Deleting Vertices To Bound Path Length

Doowon Paik+ Sudhakar Reddy++ Sartaj Sahni+
University of Florida University of Iowa University of Florida

Abstract

We examine the vertex deletion problem for weighted directed acyclic graphs (wdags). The objective is to delete the fewest number of vertices so that the resulting wdag has no path of length > δ. Several simplified versions of this problem are shown to be *NP*-hard. However, the problem is solved in linear time when the wdag is a rooted tree and in quadratic time when the wdag is a series-parallel graph.

Keywords And Phrases

Vertex deletion, directed acyclic graphs, rooted trees, series-parallel graphs, NP-hard

+ Research supported, in part, by the National Science Foundation under grants DCR-84-20935 and MIPS-86-17374.
++ Research supported, in part, by SDIO/IST Contract No. N00014-90-J-1793 managed by US Office of Naval Research.

1 Introduction

A variety of vertex deletion problems formulated on graphs and digraphs are known to be NP-hard [KRIS79]. In this paper, we propose a new formulation of the vertex deletion problem that is applicable to edge weighted directed acyclic graphs (wdag). In this formulation, we are interested in deleting the smallest number of vertices from the wdag such that the resulting wdag has no path of length $>\delta$ where δ is an input to the problem.

This problem is a natural variant of the vertex splitting problem for weighted wdags that we studied in [PAIK90]. In the vertex splitting problem, we are split the fewest number of vertices so that the resulting wdag has no path of length $>\delta$. When a vertex is split, two vertices are created. The incoming edges of the original vertex are attached to one of these and the outgoing edges to the other. The vertex splitting problem can be used to model several VLSI design problems such as the selection of flip-flops for scan paths in partial scan designs and the placement of signal boosters in lossy circuits.

The vertex deletion problem described above can be used to model certain VLSI design and communication problems. For example, suppose we have a collection of interconnected modules with the property that the flow of signals can be modeled by a wdag. The edge weights give the signal level loss incurred as the signal travels on each edge. Let δ denote the maximum permissible signal loss. Assume that it is possible to upgrade the modules and their connections so that there is effectively no signal loss on the upgraded connections (these may for e.g. be gold) and the upgraded module restores the signal level to its maximum value. To minimize the cost of the resulting circuit, it is desired to upgrade the minimum number of modules such that in the resulting circuit no signal suffers a loss $> \delta$ before being restored to its maximum level. This is equivalent to deleting the fewest number of vertices so that the resulting wdag has no path of length $> \delta$.

The terminology we shall use in this paper is developed in the next section. In this section, we also obtain some relationships between the size of an optimal solution to the vertex deletion problem and that of an optimal solution to the vertex splitting problem on the same wdag. Our NP-hard results are presented in Section 3. In Section 4, we develop a linear time algorithm for wdags that rooted trees and in Section 5, a quadratic algorithm is developed for wdags that are series-parallel graphs. We note that this quadratic algorithm is easily adapted to the vertex splitting problem on series-parallel graphs. This adaptation does not affect its complexity. Likewise, the backtracking algorithm and heuristics proposed in [PAIK90] for the vertex splitting problem may be easily adapted to the vertex deletion problem of this paper.

2 Terminology

Let $G = (V,E,w)$ be a weighted *directed acyclic graph* 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 vetex* is a vertex with zero in-degree while a *sink vetex* 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 $G–X$ 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. Figure 1(a) shows an example wdag G and Figure 1(b) shows the wdag $G–\{v_5\}$. Note that the source and sink vertex sets of G and $G–X$ may be different. 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. For the wdag of Figure 1(a), the set $X = \{v_5\}$ is a solution to the DVDP problem when $\delta = 3$.

Let G/X be the wdag that results when each vertex v in X is split into two vertices 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 2 shows the result, $G/\{v_5\}$, of splitting the vertex v_5 of the wdag of Figure 1(a). The *dag vertex splitting problem* (DVSP) is to find a least cardinality vertex set X such that $d(G/X) \le \delta$, where δ is a prespecified delay. For the wdag of Figure 1(a) and $\delta = 3$, $X = \{v_4, v_5\}$ is a solution to the DVSP problem.

Lemma 1: 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 $\delta \ge 0$.

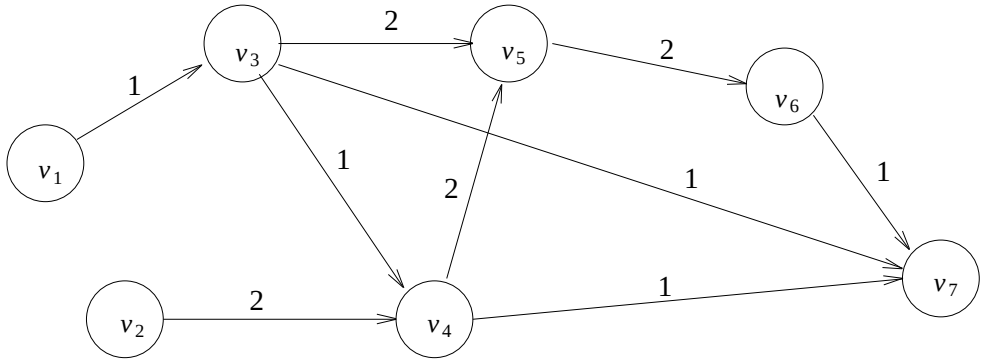
(b) The DVSP has a solution iff $\delta \ge$ MaxEdgeDelay.

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

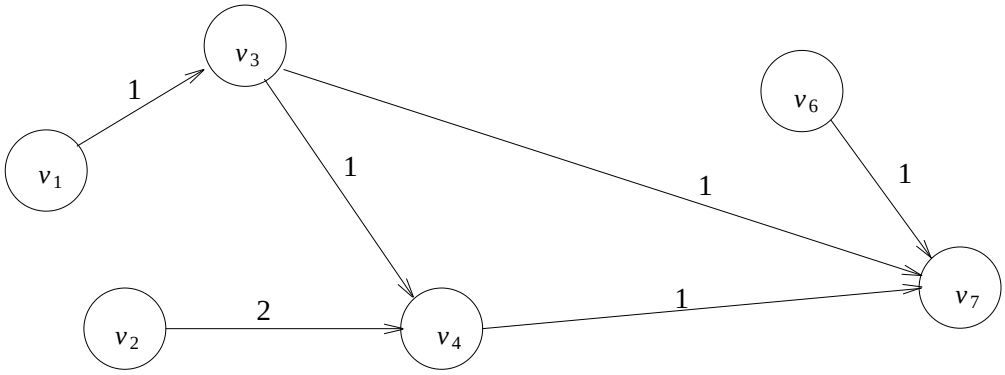
Proof:

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

(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 $\delta \ge$ MaxEdgeDelay, there is a least cardinality set X such that $d(G/X) \le \delta$.



(a)



(b)

Figure 1: DVDP example.

(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 2: 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 3. $d(G - \{v\}) = \delta$. 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. □

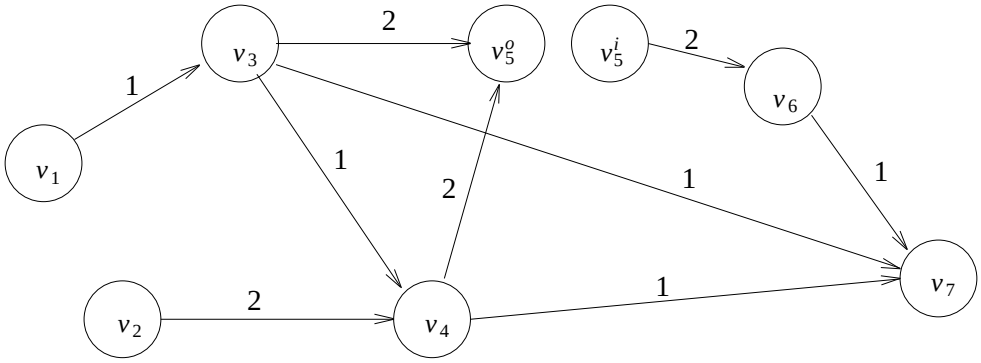


Figure 2: DVSP example.

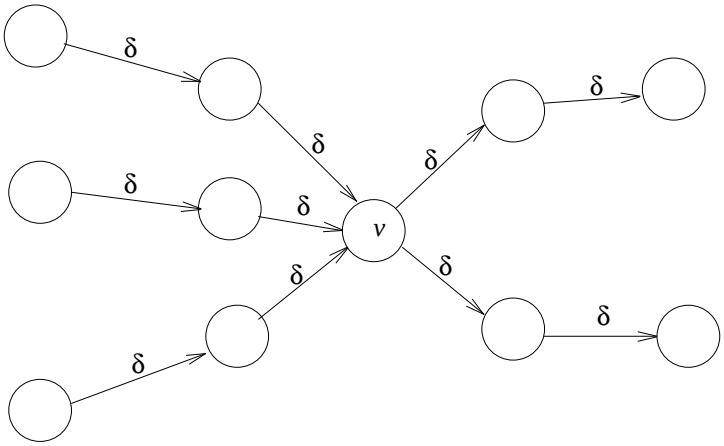


Figure 3: Construction for Lemma 2.

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 1 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. □

3 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. In this section we show that the following problems are *NP*-hard.

1. DVDP for unit weight graphs with $\delta \geq 0$.
2. DVDP for unit weight multistage graphs with $\delta \geq 2$ (in a multistage graph the vertices are divided into an ordered set of stages and each edge goes from a vertex in one stage to one in the next stage).

Since unit weight wdags are just a special case of general wdags, the results obtained imply the *NP*-hardness of the corresponding problems with the unit weight constraint removed.

3.1 Unit Weight DVDP

First, we shall show that unit weight DVDP is NP-hard when $\delta = 0$. This is done by showing that the vertex cover problem (i.e., given an undirected graph G does it contain a set, S, of at most k vertices such that each edge of G is incident to at least one vertex in S?) which is known to be NP-complete can be solved in polynomial time if there is a polynomial time algorithm for unit weight DVDP with $\delta = 0$. Next, we use this result to show that unit weight DVDP is NP-hard for every $\delta \geq 1$.

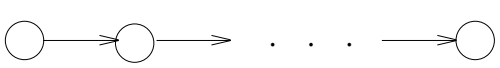
Theorem 1: Unit weight DVDP is *NP*-hard for $\delta = 0$.

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 wieght DVDP with $\delta = 0$ is *NP*-hard. □

The proof for the case $\delta \geq 1$ is obtained by showing that a polynomial time algorithm for unit weight DVDP for any fixed δ would result in a polynomial time algorithm for the case $\delta = 0$.

Theorem 2: Unit weight DVDP is *NP*-hard for $\delta \geq 1$.

Proof: Let G be an instance of unit weight DVDP with $\delta = 0$. From G construct an instance G' of unit weight DVDP for the specified value, q, of δ by attaching to each vertex of G a chain H_q of q vertices (see Figures 4 and 5). Let X be a solution to the DVDP problem on G with $\delta = 0$ and let X' be a solution to the DVDP problem on G' with $\delta = q$. We shall show that $|X| = |X'|$. Since $d(G-X) = 0$, $d(G'-X) \leq q$. Hence, $|X'| \leq |X|$. If X' contains a vertex w that is not in G then let v be the vertex of G to which the H_q that contains w was attached in the construction of G'. Let $X'' = X' - \{w\} + \{v\}$. It is easy to see that $d(G'-X'') \leq q$ and $|X''| \leq |X'|$. However, since X' is a minimal set such that $d(G'-X') \leq q$, $|X''| = |X'|$. In this way we can transform X' to X^* such that $d(G'-X^{*)} \leq q$, $|X'| = |X^*|$, and X^* consists solely of vertices of G. So, $d(G-X^{*)} = 0$. Hence, $|X'| = |X^*| \geq |X|$. Consequently, $|X'| = |X|$. From this, the observation that G' can be constructed from G in polynomial time, and the fact that unit weight DVDP with $\delta = 0$ is *NP*-hard, it follows that the unit weight DVDP with $\delta = q$ for any $q \geq 1$ is *NP*-hard. □



(a) Chain with δ vertices

(b) Schematic

Figure 4: Variable subassembly for DVDP.

3.2 DVSP For Unit Weight Multistage Graphs

A *multistage graph* is a wdag in which the vertices are partitioned into stages and each edge connects two vertices in adjacent stages. An example is given in Figure 6.

To show that DVDP for multistage graphs is *NP*-hard, we use the *NP*-hardness of the problem 2-3SAT [PAIK90] defined as:

Input: A boolean function $F = C_1\ C_2 \cdots C_n$ in n variables $x_1, x_2, \ldots, x_n$. Each clause C_i is the disjunction of either two or three literals. If $|C_i| = 2$, then both literals in C_i are either negated or unnegated. If $|C_i| = 3$, then at least one literal of C_i is unnegated and at least one is negated.

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

(a) A node in a graph (b) After attaching H_δ

Figure 5: Attaching H_δ to a node.

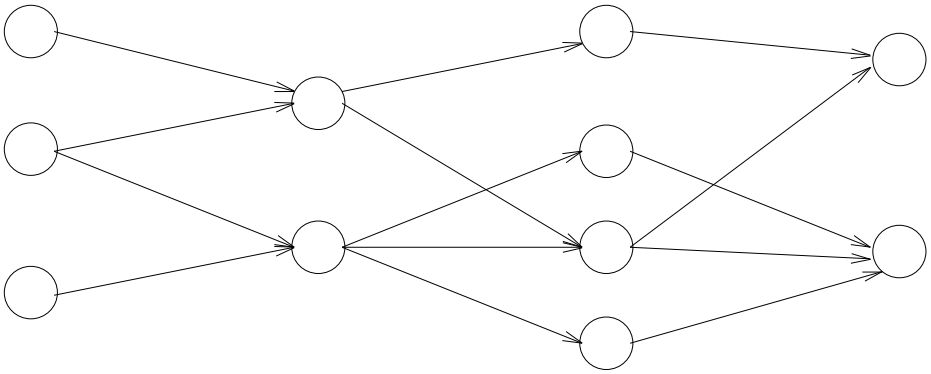


Figure 6: Example multistage graph.

For each instance F of 2-3SAT, we construct an instance G_F of the unit weight DVDP such that from the size of the solution to G_F we can determine, in polynomial time, the answer to the 2-3SAT problem for F. This construction employs two types of unit weight wdag subassemblies: variable and clause.

Variable Subassembly

The variable subassembly, $VS(i)$, for variable x_i is given in Figure 7(a). This is obtained by combining together three copies of the chain $H_{\delta-1}$ with another dag that has four vertices. Thus, the total number of vertices in the variable subassembly $VS(i)$ is $3\delta + 1$. 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(b) shows the schematic for $VS(i)$.

Clause Subassemblies

We will use two different clause subassemblies depending on the size of clause C_j in the 2-3SAT instance F. In case $|C_j| = 3$, the clause subassembly CS3(j) as in Figure 7(c) is used. The schematic for $CS3(j)$ is given in Figure 7(d). The number of vertices in $CS3(j)$ is $6\delta + 1$ and $d(CS3(j)) = 2\delta$. One may easily verify that if $|X| = 1$, then $d(CS3(j) - X) > \delta$. So, if $d(CS3(j) - X) \leq \delta$,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 $d(CS3(j) - X) \leq \delta$. If $|C_j| = 2$, the clause subassembly CS2(j) as in Figure 7(e) is used. The schematic for $CS3(j)$ is given in Figure 7(f). The number of vertices in $CS2(j)$ is $2\delta + 3$ and $d(CS2(j)) = \delta + 1$. One may easily verify that if $|X| = 1$ and $d(CS2(j) - X) \leq \delta$,then $X \subseteq \{l_{j1}, l_{j2}\}$

To construct G_F from F, we use n $VS(i)$'s, one for each variable x_i in F and q $CS2(j)$'s, one for each clause C_j in F with $|C_j| = 2$ and $m-q$ $CS3(j)$'s, one for each clause C_j in F with $|C_j| = 3$, where q is the number of the clauses of size 2. In case $|C_j| = 2$, a modified $CS2(j)$, subassembly as in Figure 8(a) is used. Connections from varible subassemblies will be made to the vertices labeled u_{j1} and u_{j2}. If $|C_j| = 3$, then a modified $CS3(j)$ is used. This modification is now described. Suppose the literals in C_j are ordered so that the unnegated ones come first. If C_j has two unnegated literals, use the clause subassembly of Figure 8(b). Otherwise, use that of Figure 8(c). There is a directed edge from vertex x_i ($\bar{x}_i$) of $VS(i)$ to vertex u_{jk} of $CS2(j)$ and $CS3(j)$ iff x_i ($\bar{x}_i$) is the k'th literal of C_j (we assume the three literals in C_j are ordered). Then the resulting graph is a multistage graph. Figure 9 gives the G_F obtained for the case $F = (x_1+\bar{x}_2+\bar{x}_4)$ $(x_2+x_3+\bar{x}_4)$ $(\bar{x}_1+\bar{x}_3)$ (x_2+x_3).

The construction of G_F can be done in polynomial time for any fixed δ.

Theorem 3: Let F be an instance of 2-3SAT and let G_F be the instance of unit weight DVDP 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 $|X| = n + 2m - q$, where m is the number of clauses in F and q is the number of two literal clauses in F.

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

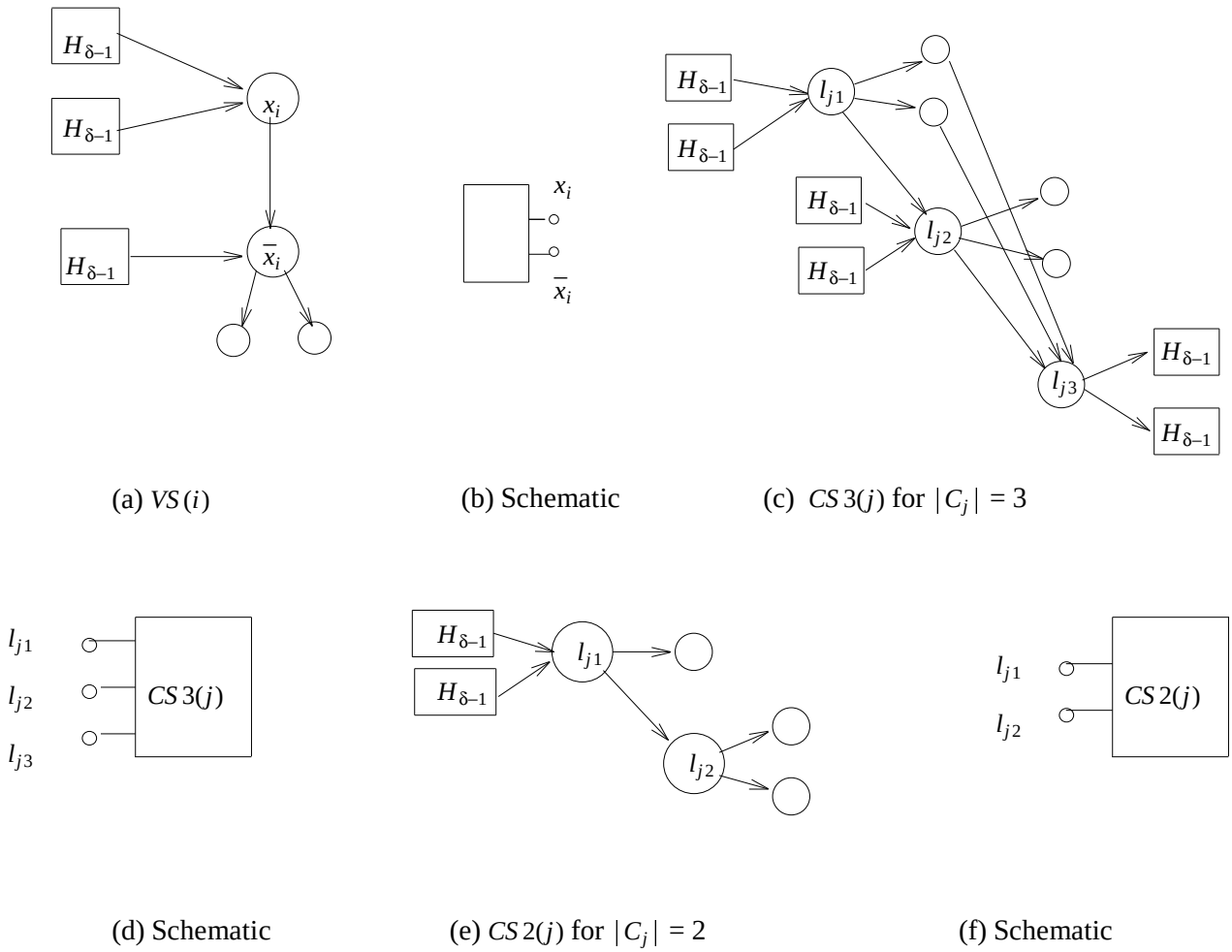
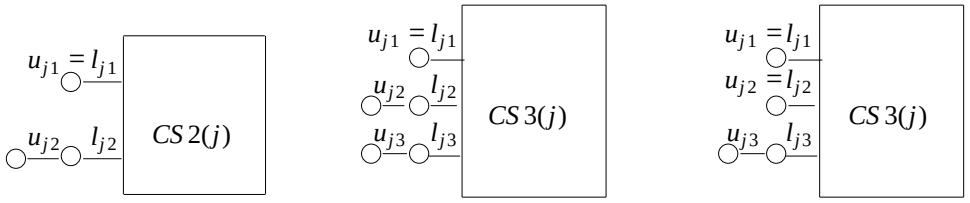


Figure 7: Subassemblies for DVDP multistage graph.

1. x_i is in X if $b_i = 1$. If $b_i = 0$, then $\bar{x}_i$ is in X.
2. From each $CS3(j)$ add exactly two of the vertices l_{j1}, l_{j2}, l_{j3} to X and from each $CS2(j)$ add exactly one of the vertices l_{j1}, l_{j2} 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 $|X| = n + 2m - q$ and that $d(G_F - X) \leq \delta$.

Next, suppose that there is an X such that $|X| = n + 2m - q = n + 2(m-q) + q$ and $d(G_F - X) \leq \delta$. From the construction of the variable and clause assemblies and from the fact that $|X| = n + 2m - q$, it follows that X must contain exactly one vertex from each of the sets $\{x_i, \bar{x}_i\}$, $1 \leq i \leq n$,



(a) $|C_j| = 2$ (b) Two unnegated literals (c) One unnegated literal

Figure 8: Modified clause subassemblies.

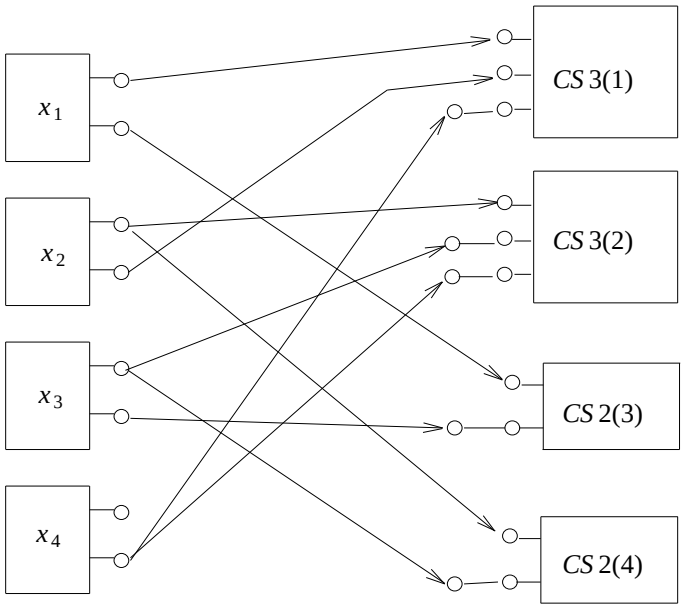


Figure 9: G_F for $F = (x_1 + \bar{x}_2 + \bar{x}_4)\,(x_2 + x_3 + \bar{x}_4)\,(\bar{x}_1 + \bar{x}_3)\,(x_2 + x_3)$.

exactly 2 from the sets $\{l_{j1}, l_{j2}, l_{j3}\}$ of each of the $m-q$ $CS3(j)$'s and exactly 1 from the sets $\{l_{j1}, l_{j2}\}$ of each of the q $CS2(j)$'s. Hence there is no i such that both $x_i \in X$ and $\overline{x}_i \in X$, there is no j for which $l_{j1} \in X$ and $l_{j2} \in X$ and $l_{j3} \in X$ if $|C_j| = 3$ and there is no j for which $l_{j1} \in X$ and $l_{j2} \in X$ if $|C_j| = 2$. Consider the Boolean assignment $b_i = 1$ iff $x_i \in X$. Suppose that $l_{jk} \notin X$ and $l_{jk} = x_i$ ($\overline{x}_i$). Since $d(G_F - X) \leq \delta$, vertex x_i ($\overline{x}_i$) must be removed as otherwise there is a source to sink path with delay greater than δ. So, x_i ($\overline{x}_i$) $\in X$ and $b_i = 1$ (0). As a result, the k'th literal of clause C_j is true. Hence, $b_1, \ldots b_n$ results in each clause having at least one true literal and F has value 1. □

4 Tree DVDP

In this section we develop a linear time algorithm for the DVDP when the wdag G is a rooted tree. The algorithm is a simple 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 $D(x)$ exceeds δ, the node x is deleted from G.

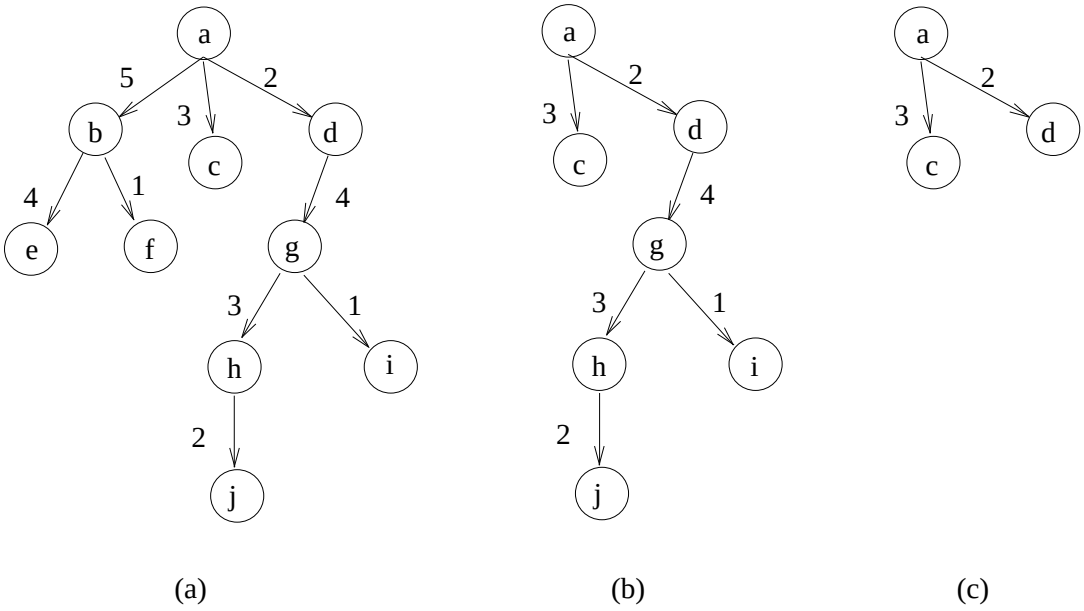


Figure 10: An example tree.

Consider the example tree of Figure 10(a) and assume $\delta = 3$. The delay, $D(x)$, for x a leaf node is 0. So, $D(x) = 0$ for $x \in \{ e , f , c , j , i \}$. In postorder, a node is visited after its children have been. When a node x is visited, its delay may be computed as:

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

So, $D(b) = 4$ and $D(h) = 2$. Since $D(b) > \delta$=3, we delete node b to get the tree of Figure 10(b). Next, $D(g) = 5$ is computed and node G is deleted resulting in the tree of Figure 10(c). $D(d) = 0$ and $D(a) = 3$. No more nodes are deleted. The formal algorithm is given in Figure 11. The algorithm assumes that X has been initialized to $\varnothing$.

```
procedure DVDP_tree(T);
   {Find minimum cardinality X such that d(T–X) ≤ δ}
   {Assume that X is initialized to ∅}
begin
   if T < > nil
   then begin
          D(T) = 0;
          for each child Y of T do
          begin
             DVDP_tree(Y);
             if Y ∉ X
             then D(T):= max {D(T), D(Y)+w(T,Y)};
          end;
          if D(T) > δ then X:=X ∪ {T};
         end;
end; {of DVDP_tree}
```

Figure 11: DVDP algorithm for trees.

Theorem 4: Procedure DVDP_tree finds a minimum cardinality X such that $d(T–X) \leq \delta$.

Proof: The proof is by induction on the number, n, of nodes in the tree T. If $n = 0$, the theorem is trivially valid. Assume this is so for $n \leq m$ where m is an arbitrary natural number. Let T be a tree with $n + 1$ nodes. Let X be the set of vertices deleted by DVDP_tree and let W be a minimum cardinality vertex set such that $d(T–W) \leq \delta$. We need to show that $|X| = |W|$. If $|X| = 0$, this is

trivially true. If $|X| > 0$, then let z be the first vertex added to X by DVDP_tree. Let T_z be the subtree of T rooted at z. As z is added to X by DVDP_tree, $D(z) > \delta$. Hence, W must contain at least one vertex u that is in T_z. If W contains more than one such u, then W cannot be of minimum cardinality as $Z = W - \{$ all such $u \} + \{z\}$ is such that $d(T-Z) \leq \delta$. Hence, W contains exactly one such u. Let $W' = W - \{u\}$. Let $T' = T - T_z$ be the tree that results from the removal of T_z from T. If there is a W^* such that $|W^*| < |W'|$ and $d(T'-W^*) \leq \delta$, then since $d(T-W^* - \{z\}) \leq \delta$, W is not a minimum cardinality deletion set for T. So, W' is a minimum cardinality vertex set such that $d(T'-W') \leq \delta$. Also, $X' = X - \{z\}$ is such that $d(T'-X') \leq \delta$ and furthermore X' is the answer produced by DVDP-tree when started with T'. Since the number of vertices in T' is less than $m + 1$, $|X'| = |W'|$. Hence, $|X| = |X'| + 1 = |W'| + 1 = |W|$. □

The complexity of procedure DVDP_tree is easily seen to be $O(n)$ where n is the number of vertices in T.

5 Series-Parallel DVDP

5.1 Definitions

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 12(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 12(d)). $G_1 G_2$ is an SPDAG.

For any SPDAG G and delay values δ, $l \leq \delta$ and $r \leq \delta$, let $S(l,r,G)$ be a minimum cardinality vertex set such that:

(C1) The length of every path in $G - S(l,r,G)$ that begins at the source of G is $\leq l$

(C2) The length of every path in $G - S(l,r,G)$ that ends at the sink of G is $\leq r$

(C3) The length of every path in $G - S(l,r,G)$ is $\leq \delta$

If $S(l,r,G)$ contains the source (sink) of G, then, by convention, l (r) has value -1. Figure 13(a) gives an example. Here, $l = 2$, $r = 1$, and $\delta = 4$. For these values of of l, r, and δ, $S = \{v_1, v_2, v_3,\}$ and $G - S(l,r,G)$ is as in Figure 13(b). One may verify that there is no smaller size vertex set that satisfies the requirements on S.

Let $f(G) = \{ (l,r, |S(l,r,G)|) \mid -1 \leq l,r \leq \delta \}$. Let (l_1, r_1, k_1) and (l_2, r_2, k_2) be two different

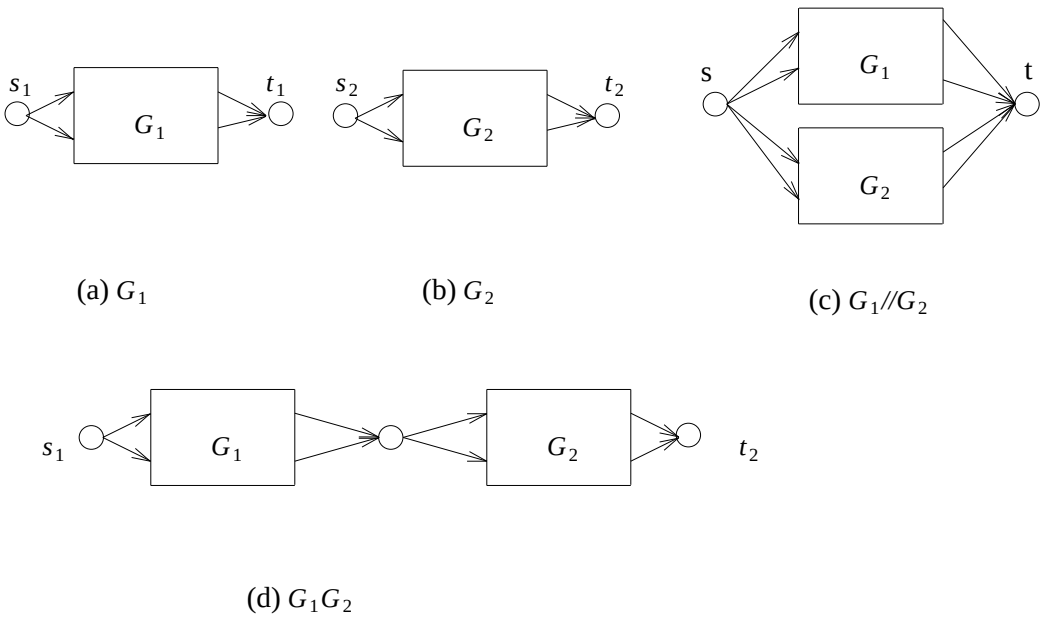


Figure 12: Series-parallel digraph.

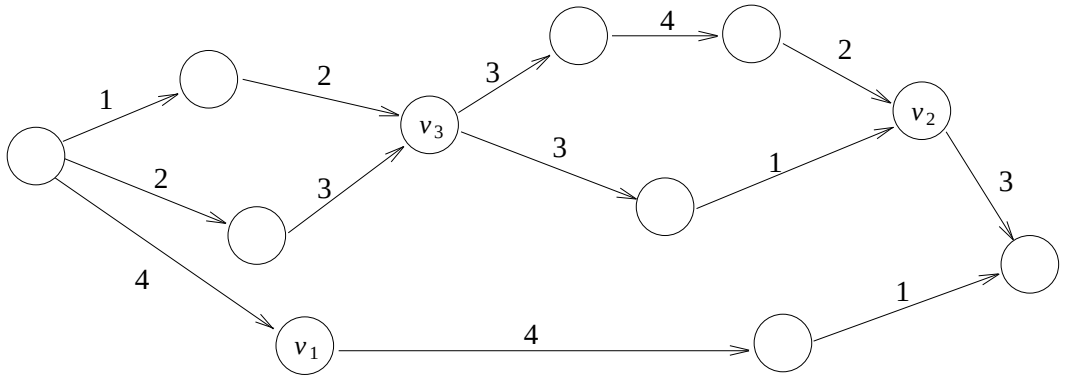
triples in $f(G)$. (l_1, r_1, k_1) dominates (l_2, r_2, k_2) iff $l_1 \le l_2$, $r_1 \le r_2$, and $k_1 \le k_2$. We shall develop an algorithm to compute the set , $F(G)$, of nondominated triples of $f(G)$. Let (l,r,z) be the triple in $F(G)$ with least third cordinate z. It is easy to see that the DVDP for G has a solution of size z. This solution may be found by modifying our algorithm so that each triple includes not just $|S(l,r,G)|$ but also $S(l,r,G)$.

Our algorithm to compute $F(G)$ is specified recursively and is based on the recursive definition of SPDAGs.

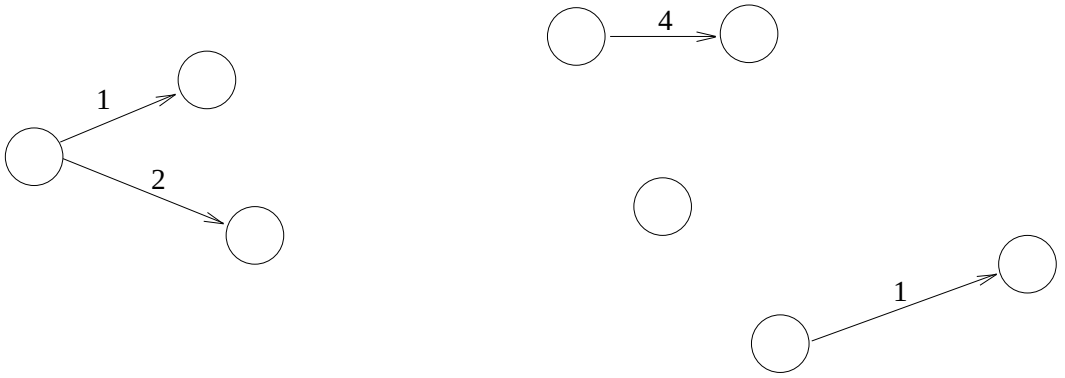
5.2 G is of the form G_1G_2

When G is the series combination of G_1 and G_2, $F(G_1G_2)$ can be obtained from $F(G_1)$ and $F(G_2)$ by using the following lemma.

Lemma 3: If $(l,r,k) \in F(G_1G_2)$, then there is an $(l_1, r_1, k_1) \in F(G_1)$ and an $(l_2, r_2, k_2) \in F(G_2)$ such that $S(l_1, r_1, G_1) = S(l,r,G_1G_2) \cap V(G_1)$, $S(l_2, r_2, G_2) = S(l,r,G_1G_2) \cap V(G_2)$, $S(l,r,G_1G_2) = S(l_1, r_1, G_1) \cup S(l_2, r_2, G_2)$, $k_1 = |S(l_1, r_1, G_1)|$, $k_2 = |S(l_2, r_2, G_2)|$, and $k = |S(l,r,G_1G_2)|$.



(a) Series-parallel graph G



(b) $G - S(l,r,G)$

Figure 13: An example

Proof: Let $A = S(l,r,G_1G_2) \cap V(G_1)$ and $B = S(l,r,G_1G_2) \cap V(G_2)$. Clearly, $S(l,r,G_1G_2) = A \cup B$. Let l_1 and r_1 be the least values such that:

a) The length of every path in $G_1 - A$ that begins at the source of G_1 is $\leq l_1$ (in case the source of G_1 is in A, let $l_1 = -1$).

b) The length of every path in $G_1 - A$ that ends at the sink of G_1 is $\leq r_1$ (in case the sink of G_1 is in A, let $r_1 = -1$).

Note that no path in $G_1 - A$ has length $> \delta$, $l_1 \leq l \leq \delta$, and $r_1 \leq \delta$. So, if $(l_1, r_1, k_1 = |A|) \notin F(G_1)$, then $F(G_1)$ must contain a triple that dominates (l_1, r_1, k_1). Let (l', r', k') be such a triple. If $r_1 = -1$, $r' = r_1$, sink(G_1) = source$(G_2) \in A$, and sink$(G_1) \in C = S(l', r', G_1)$. So, $|C \cup B| = k' + |B| - 1$ and $|A \cup B| = |A| + |B| - 1$. If $r_1 \neq -1$, sink$(G_1) \notin A$ and source$(G_2) \notin B$. So, $|C \cup B| = k' +$

$|B|$ and $|A \cup B| = |A| + |B|$. Furthermore, let l^* and r^* be the least values that satisfy a) and b) above with G_1 replaced by G_1G_2, l_1 by l^*, r_1 by r^*, and A by $C \cup B$. It is easy to see that no path in $G_1G_2 - C \cup B$ has length $> \delta$, $l^* \le l$, $r^* \le r$ and $|C \cup B| \le |A \cup B|$. So, $(l^*,r^*, |C \cup B|)$ dominates (l,r,k) and so $(l,r,k) \notin F(G_1G_2)$. A contradiction. Hence, $(l_1,r_1,k_1) \in F(G_1)$. We can show $(l_2,r_2,k_2=|B|) \in F(G_2)$ similarly. □

Lemma 3 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 considering every pair (l_1,r_1,k_1) and (l_2,r_2,k_2) of triples such that :

1) $(l_1,r_1,k_1) \in F(G_1)$ and $(l_2,r_2,k_2) \in F(G_2)$

2) $S(l,r,G_1G_2) = S(l_1,r_1,G_1) \cup S(l_2,r_2,G_2)$ satisfies C1, C2 and C3 and l and r are as small as possible. The triple $(l,r, |S(l,r,G_1G_2)|)$ is added to Z

Step2: Triples in Z that are dominated by other triples in Z are deleted.

To implement step1, we need to consider several cases for (l_1,r_1,k_1) and (l_2,r_2,k_2). These are considered below. To determine which case applies, the case conditions are tested sequentially in the order listed and the first one satisfied is the one that applies.

Let $A = S(l_1,r_1,G_1)$, $B = S(l_2,r_2,G_2)$ and $C = A \cup B$.

Case 1 : $l_2 = r_1 = -1$

Now, $(l_1,r_2,|C|)$ satisfies conditions C1 – C3. Since $A \cap B = sink(G_1) = source(G_2)$, $|C| = k_1 + k_2 - 1$. Hence, $(l_1,r_2,k_1 + k_2 - 1)$ is a candidate for $F(G_1G_2)$.

Case 2 : $k_1 = k_2 = 0$

When $k_1 = k_2 = 0$, $A = B = C = \varnothing$. Since G_1 and G_2 are connected graphs, we have $l_1 = r_1$ and $l_2 = r_2$ and $d(G_1G_2) = l_1 = l_2$. So, if $l_1 + l_2 > \delta$, C does not satisfy conditions C1, C2, and C3. On the other hand, if $l_1 + l_2 \le \delta$, C satisfies these conditions and $(l_1 + l_2, l_1 + l_2, 0)$ is a candidate for $F(G_1G_2)$.

Case 3 : $k_1 = 0, k_2 \ne 0$

If $l_2 = -1$, then since $A = \varnothing$, $source(G_2) = sink(G_1) \notin A$ and $A \cup B$ is not a valid vertex deletion set for G_1G_2. So, assume $l_2 \ne -1$. Now, in $G_1G_2 - C$ the longest path from the source has length $r_1 + l_2 = l_1 + l_2$. So, if $l_1 + l_2 > \delta$, condition C1 is not satisfied. Hence, we require $l_1 + l_2 \le \delta$. In this case, $(l_1 + l_2, r_2, k_2)$ is a candidate for $F(G_1G_2)$.

Case 4 : $k_1 \ne 0, k_2 = 0$

This is similar to Case 3. For C to be a valid deletion set, we require $r_1 \ne -1$ and $r_1 + r_2 \le \delta$. When this is the case, $(l_1, r_1 + r_2, k_1)$ is a candidate for $F(G_1G_2)$.

Case 5 : everything else

Now, $k_1 \ne 0$ and $k_2 \ne 0$. If $r_1 \ne -1$, then $l_1 \ne -1$ as otherwise we are in Case 1. However, $r_1 =$

-1 implies $sink(G_1) \in A$ and $l_2 \neq -1$ implies $source(G_2) \notin B$. Since $sink(G_1) = source(G_2)$, $C = A \cup B$ is not a valid deletion set for G_1G_2. So, if $r_1 = -1$ we must have $l_2 = -1$ and be in Case 1. For Case 5, therefore, we may assume $r_1 \neq -1$ and $l_2 \neq -1$. Now, the longest path in $G_1G_2 - C$ from the source vertex has length l_1 and the longest path to the sink has length r_2. If $l_2 + r_1 \leq \delta$, C satisfies $C1$–$C3$ and $(l_1, r_2, k_1 + k_2)$ is a candidate for $F(G_1G_2)$.

The above discussion results in the algorithm of Figure 14. The notation $F(G_1G_2) \oplus (l,r,k)$ means the set that results from adding (l,r,k) to $F(G_1G_2)$ and deleting all dominated triples.

$F(G_1G_2) = \emptyset$
for every $(l_1, r_1, k_1) \in F(G_1)$ **do**
  for every $(l_2, r_2, k_2) \in F(G_2)$ **do**
    case
      : $l_2 = r_1 = -1$:
        $F(G_1G_2) = F(G_1G_2) \oplus (l_1, r_2, k_1 + k_2 - 1)$
      : $k_1 = k_2 = 0$ and $l_1 + l_2 \leq \delta$:
        $F(G_1G_2) = F(G_1G_2) \oplus (l_1 + l_2, l_1 + l_2, 0)$
      : $k_1 = 0$ and $k_2 \neq 0$ and $l_2 \neq -1$ and $l_1 + l_2 \leq \delta$:
        $F(G_1G_2) = F(G_1G_2) \oplus (l_1 + l_2, r_2, k_2)$
      : $k_1 \neq 0$ and $k_2 = 0$ and $r_1 \neq -1$ and $r_1 + r_2 \leq \delta$:
        $F(G_1G_2) = F(G_1G_2) \oplus (l_1, r_1 + r_2, k_1)$
      : $k_1 \neq 0$ and $k_2 \neq 0$ and $r_1 \neq -1$ and $l_2 \neq -1$ and $r_1 + l_2 \leq \delta$:
        $F(G_1G_2) = F(G_1G_2) \oplus (l_1, r_2, k_1 + k_2)$
    endcase

Figure 14: Algorithm for $F(G_1G_2)$.

5.3 G is of the form $G_1 // G_2$

Lemma 4: If $(l,r,k) \in F(G_1 // G_2)$, then there is an $(l_1, r_1, k_1) \in F(G_1)$ and an $(l_2, r_2, k_2) \in F(G_2)$ such that $S(l_1, r_1, G_1) = S(l,r,G_1 // G_2) \cap V(G_1)$, $S(l_2, r_2, G_2) = S(l,r,G_1 // G_2) \cap V(G_2)$, $S(l,r,G_1 // G_2) = S(l_1, r_1, G_1) \cup S(l_2, r_2, G_2)$, $k_1 = |S(l_1, r_1, G_1)|$, $k_2 = |S(l_2, r_2, G_2)|$, and $k =$

$|S(l,r,G_1 // G_2)|$.

Proof: Similar to that of Lemma 3. □

For every pair $(l_1, r_1, k_1) \in F(G_1)$ and $(l_2, r_2, k_2) \in F(G_2)$, we need to consider the following cases. A, B and C are defined as in the discussion for $F(G_1 G_2)$.

Case 1 : $l_1 = l_2 = -1$, $r_1 = r_2 = -1$

Since $A \cap B = \{ source(G_1) , sink(G_1) \} = \{ source(G_2) , sink(G_2) \}$, $|C| = k_1 + k_2 - 2$ and $(-1, -1, k_1 + k_2 - 2)$ is a candidate for $F(G_1 // G_2)$.

Case 2 : $l_1 = l_2 = -1$

Now, $|C| = k_1 + k_2 - 1$. If $r_1 = -1$ or $r_2 = -1$, then the resulting triple is $(-1, -1, k_1 + k_2 - 1)$ which is dominated by a triple from Case 1. So, we need only consider the case $r_1 \geq 0$ and $r_2 \geq 0$. In this case the candidate triple for $F(G_1 // G_2)$ is $(-1 , \max\{r_1, r_2\}, k_1 + k_2 - 1)$.

Case 3 : $r_1 = r_2 = -1$

This is similar to Case 2 and we require $l_1 \geq 0$ and $l_2 \geq 0$. The candidate triple for $F(G_1 // G_2)$ is $(\max\{l_1, l_2\}, -1, k_1 + k_2 - 1)$.

Case 4 : everything else

We need only consider the case when $l_1 \neq -1$, $l_2 \neq -1$, $r_1 \neq -1$ and $r_2 \neq -1$. Now, $(\max\{l_1, l_2\}, \max\{r_1, r_2\}, k_1 + k_2)$. is a candidate for $F(G_1 // G_2)$. The resulting algorithm is given in Figure 15.

5.4 G is a chain

If G has only two vertices u and v, then $F(G)$ is : $\{(-1,-1,2), (-1,0,1), (0,-1,1)\}$ when $w(u,v) > \delta$ and $\{(-1,-1,2), (-1,0,1), (0,-1,1), (w(u,v), w(u,v), 0)\}$ when $w(u,v) \leq \delta$.

If G has more than two vertices, then G is of the form $G_1 G_2$ and $F(G)$ may be obtained using the algorithm for $F(G_1 G_2)$ (Figure 14).

A simpler algorithm for $F(G)$ results by considering procedure Greedy (Figure 16). This procedure considers a chain $C = i , i+1 , \cdots , j$ and determines a minimum cardinality vertex set X such that $C - X$ has no path of length $> \delta$. The algorithm for $F(G)$ when G is the chain 1,2, ... ,n is given in Figure 17. This algorithm tries all candidates for the leftmost vertex in $S(l,r,G)$. Once a candidate is selected, the remaining vertices in $S(l,r,G)$ may be obtained using procedure Greedy.

5.5 Analysis

The correctness of the algorithms for $F(G_1 G_2)$, $F(G_1 // G_2)$, and $F(chain)$ follows from the discussions in Sections 5.2, 5.3, and 5.4. For the complexity analysis, we need the following lemmas.

$F(G_1 /\!/ G_2) = \varnothing$
for every $(l_1, r_1, k_1) \in F(G_1)$ **do**
  for every $(l_2, r_2, k_2) \in F(G_2)$ **do**
    case
      : $l_1 = l_2 = -1$ and $r_1 = r_2 = -1$:
        $F(G_1 /\!/ G_2) = F(G_1 /\!/ G_2) \oplus (-1, -1, k_1 + k_2 - 2)$
      : $l_1 = l_2 = -1$ and $r_1 \neq -1$ and $r_2 \neq -1$:
        $F(G_1 /\!/ G_2) = F(G_1 /\!/ G_2) \oplus (-1, \max\{r_1, r_2\}, k_1 + k_2 - 1)$
      : $r_1 = r_2 = -1$ and $l_1 \neq -1$ and $l_2 \neq -1$:
        $F(G_1 /\!/ G_2) = F(G_1 /\!/ G_2) \oplus (\max\{l_1, l_2\}, -1, k_1 + k_2 - 1)$
      : $l_1 \neq -1$ and $l_2 \neq -1$ and $r_1 \neq -1$ and $r_2 \neq -1$:
        $F(G_1 /\!/ G_2) = F(G_1 /\!/ G_2) \oplus (\max\{l_1, l_2\}, \max\{r_1, r_2\}, k_1 + k_2)$
    endcase

Figure 15: Algorithm for $F(G_1 /\!/ G_2)$.

Lemma 5: Let G be an arbitrary SPDAG. G is formed by performing some number of parallel and series combinations on directed chains. Let p be the number of parallel combinations performed. Let $(l,r,k) \in F(G)$ and $n = |V(G)|$. The number of different values for l is at most $n + p + 1$.

Proof: Let L be the set of different values for l. We shall show that $|L - \{0, -1\}| \leq n + p - 1$. The proof is by induction on n. When $n = 2$, $p = 0$ and $|L - \{0, -1\}| \leq 1$. Assume that $|L - \{0, -1\}| \leq n + p - 1$ for all SPDAG's with $n \leq m$ vertices. We need to show that $|L - \{0, -1\}| \leq m + p$ for all SPDAG's with $m + 1$ vertices. Let G be one such SPDAG. If G is a chain, then $|L - \{0, -1\}| \leq m$ and $p = 0$. If $G = G_1 G_2$, then let n_1 and n_2, respectively, be the number of vertices in G_1 and G_2 and let p_1 and p_2 be the number of parallel combinations in G_1 and G_2. Clearly, $p = p_1 + p_2$, $n = n_1 + n_2 - 1$, $n_1 \leq m$, and $n_2 \leq m$. From the induction hypothesis it follows that:

```
procedure Greedy (i, j, l, r, X);
   { Find least cardinality X such that no path in the graph C–X has
     length > δ. C is the chain  i, i+1, ··· ,j,  l is the length of the
     leftmost component of C–X and r that of the rightmost component }
begin
    X = ∅ ;  length := 0;  l:= – 1;  r:= – 1;
    for a = i + 1 to j do
        if length + w(a–1,a) > δ
                then begin
                        X:= X ∪ {a};
                        if l := – 1 then l := length;
                        if a < j  then length := – w(a,a+1);
                     end;
        else length:= length + w(a–1,a);
    r:= length;
    if X = ∅ then l:= r;
end;
```

Figure 16: Algorithm for least X.

$$
\begin{aligned}
|\, L - \{0,-1\}\,| \;&\le\; |\, L_1 - \{0,-1\}\,| \;+\; |\, L_2 - \{0,-1\}\,| \\
&\quad (\, L_1 \text{ and } L_2 \text{ are, respectively, the number of different } l \\
&\qquad \text{values in } F(G_1) \text{ and } F(G_2)\,) \\
&\le n_1 + p_1 - 1 + n_2 + p_2 - 1 \\
&= (n_1 + n_2 - 1) + (p_1 + p_2) - 1 \\
&= n + p - 1.
\end{aligned}
$$

If $G = G_1 // G_2$, then $p = p_1 + p_2 + 1$ and $n = n_1 + n_2 - 2$. If $n_1 > 2$ and $n_2 > 2$, then $n_1 \le m$ and $n_2 \le m$ and the induction hypothesis yields:

```
procedure Chain(n);
   { Obtain F(G) for G an n vertex chain 1,2, ... n }
  begin
    { special cases }
    Greedy (2, n–1, l, r, X)
    F(G):= (– 1, – 1, |X| + 2);
    Greedy (1, n–1, l, r, X)
    F(G):= F(G) ⊕ (l, – 1, |X| + 1);
    Greedy (2, n, l, r, X)
    F(G):= F(G) ⊕ (– 1, r, |X| + 1);

    { remaining cases }
    l:= 0;  a:=2;
    while l ≤ δ and a < n do
      {a is leftmost vertex in S(l,r,G)}
    begin
        Greedy (a+1, n, b, r, X);
        F(G):= F(G) ⊕ (l, r, |X| + 1);
        l:= l + w(a–1,a);
        a:= a +1
    end;
    if a = n and  l + w(a–1,a) ≤ δ
    then
    F(G):= F(G) ⊕ (l + w(a–1,a), l + w(a–1,a), 0);
  end;
```

Figure 17: Algorithm for $F(G)$ wnen G is a chain.

$$
\begin{aligned}
|\, L - \{0,-1\} \,| \;&\le\; |\, L_1 - \{0,-1\} \,| \;+\; |\, L_2 - \{0,-1\} \,| \\
&\le n_1 + p_1 - 1 + n_2 + p_2 - 1 \\
&= (n_1 + n_2 - 2 + 2) + (p_1 + p_2 + 1 - 1) - 2 \\
&= n + 2 + p - 1 - 2
\end{aligned}
$$

$= n + p - 1$.

If $n_1 = 2$, then $n_2 = n = m + 1$ and the induction hypothesis cannot be used on G_2. If $G_2 = G_3G_4$, then the above proof for a series combination implies that $| L_2 - \{0, -1\} | \leq n_2 + p_2 - 1$. If $G_2 = G_3 // G_4$, then from the definition of an SPDAG and the fact that $n_1 = 2$, it follows that $n_3 > 2$ and $n_4 > 2$. Hence, the preceding proof for this case implies that $| L_2 - \{0, -1\} | \leq n_2 + p_2 - 1$. So, regardless of whether $G_2 = G_3G_4$ or $G_3 // G_4$, $| L_2 - \{0, -1\} | \leq n_2 + p_2 - 1$. Hence, $| L - \{0, -1\} | \leq n_1 + p_1 - 1 + n_2 + p_2 - 1 = n + p - 1$. The proof for the case $n_2 = 2$ is similar. □

Corollary 2: The number of different values for l is at most $2n$.

Proof: Follows from Lemma 5 and the fact that $p \leq n - 1$. □

Lemma 6: Let (l, r_1, k_1) and (l, r_2, k_2) be two different triples in $F(G)$. The following are true:

(a) $|k_1 - k_2| = 1$

(b) $r_1 = -1$ or $r_2 = -1$

(c) If $r_1 = -1$, then $r_2 \geq 0$ and $k_1 = k_2 + 1$

(d) If $r_2 = -1$, then $r_1 \geq 0$ and $k_2 = k_1 + 1$

Proof: If $k_1 = k_2$, then one of the triples dominates the other and so one should not be in $F(G)$. Assume that $k_1 > k_2$. Since neither triple dominates the other, it must be that $r_1 < r_2$ and $r_2 \geq 0$. Hence, $sink(G) \notin S(l, r_2, G)$. By adding $sink(G)$ to $S(l, r_2, G)$ we get a triple $(l, -1, k_2 + 1)$ which dominates (l, r_1, k_1) unless $r_1 = -1$ and $k_1 = k_2 + 1$. A similar reasoning for the case $k_1 < k_2$ requires $r_2 = -1$ and $k_2 = k_1 + 1$. □

Lemma 7: $|F(G)| \leq 2n + 2$ where $n = |V(G)|$.

Proof: $F(G)$ contains one triple of each of the types : $(-1, -1, k_1)$; $(-1, r, k_2)$, $r \geq 0$; and $(l, -1, k_3)$, $l \geq 0$. Note that $k_2 = k_3 = k_1 - 1$ and that these are the only triples in $F(G)$ that have a first or second coordinate value of -1. From Corollary 2, there are at most $2n$ distinct first coordinate values in $F(G)$ and at most $2n - 1$ of these are ≥ 0. From Lemma 6, corresponding to each of these at most $2n - 1$ values there is only one second coordinate value that is ≥ 0. Hence, there are at most $2n - 1$ triples in $F(G)$ for which neither the first nor the second coordinate is -1. So, the total number of triples in $F(G)$ is at most $2n + 2$. □

Lemma 8: Let (l_1, r, k_1) and (l_2, r, k_2) be two different triples in $F(G)$. The following are true:

(a) $|k_1 - k_2| = 1$

(b) $l_1 = -1$ or $l_2 = -1$

(c) If $l_1 = -1$, then $l_2 \geq 0$ and $k_1 = k_2 + 1$

(d) If $l_2 = -1$, then $l_1 \geq 0$ and $k_2 = k_1 + 1$

Proof: Similar to that of Lemma 6. □

Lemma 9: Let (l_1, r_1, k_1) and (l_2, r_2, k_2) be two triples in $F(G)$. Then, $|k_1 - k_2| \leq 2$.

Proof: $F(G)$ contains a triple $(-1, -1, k)$. If $k_1 < k - 2$, then $S(l_1, r_1, G) \cup \{ source(G), sink(G) \}$ results in a triple $(-1, -1, k_1+2)$ which dominates $(-1, -1, k)$. So, $k_1 \geq k - 2$. Similarly, $k_2 \geq k - 2$. Also, $k_1 \leq k$ and $k_2 \leq k$. Hence, $|k_1 - k_2| \leq 2$. □

Lemma 10: Let (l_1, r_1, k_1) and (l_2, r_2, k_2) be two triples in $F(G)$ such that $l_1, r_1, l_2, r_2 \neq -1$. If $k_1 < k_2$, then $l_1 > l_2$ and $r_1 > r_2$.

Proof: Assume that $k_1 < k_2$ and $l_1 \leq l_2$. If $(l_1, -1, k_1+1) \in F(G)$, then $(l_1, -1, k_1+1)$ dominates (l_2, r_2, k_2) and so $(l_2, r_2, k_2) \notin F(G)$. A contradiction. If $(l_1, -1, k_1+1) \notin F(G)$, then $(l_1, -1, k_1+1)$ is dominated by some triple in $F(G)$ and this triple dominates (l_2, r_2, k_2). So, $(l_2, r_2, k_2) \notin F(G)$. A contradiction. Hence, if $k_1 < k_2$, then $l_1 > l_2$. Similarly, if $k_1 < k_2$, then $r_1 > r_2$. □

Lemma 10 suggests organizing $F(G)$ into three lists $F(G)\cdot A$, $F(G)\cdot B$, and $F(G)\cdot C$. $F(G)\cdot A$ contains the triples with first or second coordinate equal to -1. $|F(G)\cdot A| \leq 3$. Let $(-1, -1, k)$ be a triple in $F(G)$. Then $F(G)\cdot B$ contains all triples with third coordinate $k - 1$ and $F(G)\cdot C$ contains those with third coordinate $k - 2$. The lists B and C are kept in increasing order of their first coordinate. This simultaneously results in the lists being in decreasing order of their second coordinate.

To compute $F(G_1 G_2)$ we consider triple pairs t_1, t_2 such that $t_1 \in F(G_1)\cdot X$ and $t_2 \in F(G_2)\cdot Y$ where $X, Y \in \{ A, B, C \}$. This can be done in the order $XY = AA, AB, AC, BA, BB, BC, CA, CB, CC$. Suppose we are considering the case $XY = BB$ and $k_1 \neq 0$ and $k_2 \neq 0$. Example $F(G_1)\cdot B$ and $F(G_2)\cdot B$ are given in Figure 18. This figure only shows the l and r values of each triple as the k values are the same for all triples in $F(G_1)\cdot B$ and also for all in $F(G_2)\cdot B$. Assume that $\delta = 10$. Combining $t_1 \in F(G_1)\cdot B$ and $t_2 \in F(G_2)\cdot B$ is feasible only if $r_1 + l_2 \leq \delta$. The resulting triple is $(l_1, r_2, k_1+k_2=5)$. Start two pointers i and j at 1. Pointer i moves through $F(G_1)\cdot B$ and pointer j moves through $F(G_2)\cdot B$. Since the triples i and j point to have $r_1 + l_2 = 4 + 10 = 14 > \delta$, this is an infeasible pair. Also, since r values increase as you go down $F(G_1)\cdot B$, none of the remaining triples of $F(G_1)\cdot B$ can combine with the j'th triple of $F(G_2)\cdot B$. So, we advance j to the next triple of $F(G_2)\cdot B$. Again, $r_1 + l_2 > \delta$ and j is advanced further. When $j = 4$, $r_1 + l_2 = 8$ and we get the first feasible pair t_1, t_2 which results in (10, 8, 5). Since $r_1 + l_2 < \delta$, t_2 may combine with other triples

	$k_1 = 2$	$k_2 = 3$	$k = 5$
1	(10, 4) $\leftarrow i$	$j \rightarrow$ (10, 5)	(6, 8)
2	(8, 5)	(9, 6)	(3, 10)
3	(6, 6)	(7, 7)	
4	(3, 8)	(4, 8)	
5	(2, 9)	(3, 9)	
6	(1,10)	(2,10)	
	$F(G_1)\cdot B$	$F(G_2)\cdot B$	G_1G_2

Figure 18: Example $F(G_1)\cdot B$ and $F(G_2)\cdot B$

of $F(G_1)\cdot B$. i is advanced and we get $r_1 + l_2 = 5 + 4 = 9 < \delta$. The new triple is (8, 8, 5), which dominates the previous triple (10, 8, 5). i is advanced again and we get $r_1 + l_2 = 6 + 4 = 10$. This results in the triple (6, 8, 5) which dominates (8, 8, 5). If i is advanced to 4, the t_1 t_2 combination is infeasible and so (8, 8, 5) cannot be dominated by any other combination from $F(G_1)\cdot B$ and $F(G_2)\cdot B$. (6, 8, 5) is entered into an output table. Now, $i = 4$ and $j = 4$ and t_1 t_2 is infeasible. j is advanced to 5 and then to 6. At this point, $r_1 + l_2 = 8 + 2 = 10$. If i is advanced to 5, t_1 t_2 is infeasible. So, (3, 10, 5) is not dominated by other triples and added to the output. Now $i = 5$ and $j = 6$. j cannot be advanced further and we are done combining $F(G_1)\cdot B$ and $F(G_2)\cdot B$. The time required is $O(\,|F(G_1)\cdot B| + |F(G_2)\cdot B|\,)$ as the i and j pointers advance in one direction only and following each computation of $r_1 + l_2$ either i or j advances.

All other pairs XY can similarly be handled in time $O(\,|F(G_1)\cdot X| + |F(G_2)\cdot Y|\,)$. The outputs from each XY pair can finally be merged to get $F(G_1G_2)\cdot A$, $F(G_1G_2)\cdot B$, and $F(G_1G_2)\cdot C$ in linear time. From Lemma 7, $|F(G_1G_2)| \leq 2n + 2$, $|F(G_1)| \leq 2n_1 + 2$, $|F(G_2)| \leq 2n_2 + 2$ where $n_1 = |V(G_1)|$, $n_2 = |V(G_2)|$, and $n = n_1 + n_2 - 1 = |V(G_1G_2)|$. Hence, the time to compute $F(G_1G_2)$ given $F(G_1)$ and $F(G_2)$ is $O(n)$.

Suppose we are computing $F(G_1 /\!/ G_2)$. Consider combining pairs t_1 , t_2 such that $XY = BB$ and $F(G_1)\cdot B$ and $F(G_2)\cdot B$ are as in Figure 18. For each $t_1 \in F(G_1)\cdot B$ we find a triple $t_2 \in F(G_2)\cdot B$ such that $\max\{\, l_1, l_2 \,\} = l_1$ and $\max\{\, r_1, r_2 \,\}$ is as small as possible. For this, we use pointers i

and j as in the computation of $F(G_1G_2)$. Initially, $i = j = 1$. Now, $t_1 = (10, 4, 2)$ and $t_2 = (10, 5, 3)$. Since $l_2 \le l_1$, $t_1\ t_2$ results in the triple (10, 5, 5). $\max\{ r_1, r_2 \}$ cannot be made any smaller by changing t_2 as the remaining r values in $F(G_2)\cdot B$ are larger than 5. Next consider $i = 2$. Now, $t_1 = (8, 5, 5)$ and $t_2 = (10, 5, 5)$. Since $l_2 > l_1$, we advance j to the first triple of $F(G_2)\cdot B$ for which $l_2 \le l_1$. This results in $j = 3$ and $t_2 = (7, 7, 5)$. The created triple is (8, 7, 5). Again, r cannot be reduced by considering another t_2 in $F(G_2)\cdot B$. Next, we consider $i = 3$, $t_1 = (6, 6, 2)$. $l_2 > l_1$ and j is advanced to 4. This gives the triple (6, 8, 5). When $i = 4$, j is advanced to 5 to get the triple (3, 9, 5). Next, $i = 5$ and j is advanced to 5 to get (2, 10, 5). When $i = 6$, no triple is created. The triples created, in order, are (10, 5, 5), (8, 7, 5), (6, 8, 5), (3, 9, 5), and (2, 10, 5).

Next, for each $t_2 \in F(G_2)\cdot B$ we find a triple $t_1 \in F(G_1)\cdot B$ such that $\max\{ l_1, l_2 \} = l_2$ and $\max\{ r_1, r_2 \}$ is as small as possible. This results in the triples (10, 5, 5), (9, 6, 5), (7, 7, 5), (4, 8, 5), (3, 9, 5), (2, 10, 5). Finally, the two sets of triples are merged. During the merge, dominated triples are eliminated. We get (10, 5, 5), (9, 6, 5), (7, 7, 5), (4, 8, 5), (3, 9, 5), and (2, 10, 5).

The time required is $O(\ |F(G_1)\cdot B| + |F(G_2)\cdot B|\ )$. As in the case of $F(G_1G_2)$, the remaining cases can also be handled in linear time and the total time to compute $F(G_1 /\!/ G_2)$ from $F(G_1)$ and $F(G_2)$ is $O(n)$. For a chain, $F(G)$ is easily computed in $O(n)$ time. Since an n vertex SPDAG involves fewer than n series and parallel subgraph combinations, $F(G)$ can be computed in $O(n^2)$ time using the method described above for $F(G_1 /\!/ G_2)$ and $F(G_1G_2)$.

6 Conclusions

We have shown that the unit weight DVDP problem is *NP*-hard for every δ , $\delta \ge 0$. Furthermore the problem remains *NP*-hard for $\delta \ge 2$ when the wdags are restricted to be unit weight multistage graphs. For trees we have developed a linear time DVDP algorithm and for series-parallel graphs with n vertices we have developed an $O(n^2)$ time algorithm. Our algorithm for series-parallel graphs is easily adapted to solve the wdag vertex splitting problem [PAIK90] for series-parallel graphs in $O(n^2)$ time. For general graphs, the heuristics proposed in [PAIK90] for the vertex splitting problem may be adapted to the DVDP problem.

7 References

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

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

[HORO90] E. Horowitz, and S. Sahni, "Fundamentals of Data Structures in Pascal", Computer Science Press, Maryland, 1990.

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

[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, 1990.