Upgrading Vertices In Trees, Series-Parallel Digraphs And General Series-Parallel Digraphs To Bound Path Length+

Doowon Paik
University of Minnesota

Sartaj Sahni
University of Florida

Abstract

We consider trees, series-parallel digraphs, and general series-parallel digraphs that have vertex weights and delays. The length/delay of a path is the sum of the delays on the path. We show that minimal weight vertex subsets X such that the length of the longest path is bounded by a given value δ when all vertices in X are upgraded to have delay 0 can be found in pseudo polynomial time. In case all delays are unit or all weights are unit, our algorithms have a quadratic complexity. For the case of trees with unit weights and unit delays, we develop a linear time algorithm.

Keywords And Phrases

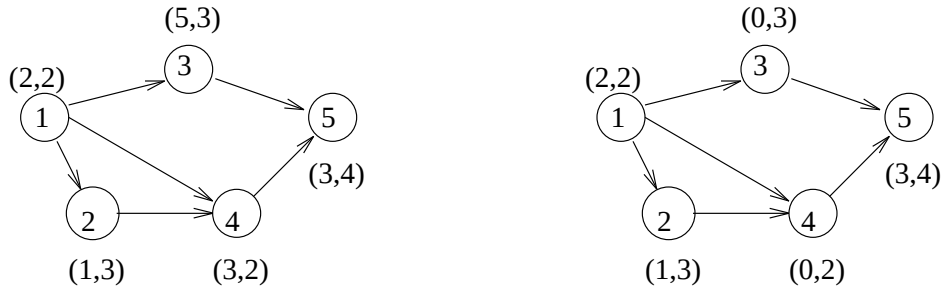
Vertex upgrading, bounding path length, trees, series-parallel digraphs, general series-parallel digraphs

+ Research supported, in part, by the National Science Foundation under grant MIP 86-17374.

1. Introduction

Directed acyclic graphs (dags) can be used to model circuits. Each vertex of the dag has a delay associated with. The delay of any signal path is the sum of the delays of the vertices on the path. The delay of a circuit is the length of the longest path in the dag. In an effort to reduce circuit delay, we may attempt to upgrade some of the dag vertices. Upgrading a vertex reduces its delay but comes at some cost. In this paper we consider an extreme form of upgrade that results in the vertex delay being negligible (i.e., zero) relative to the delay before upgrading. For each vertex v of the dag $G = (V, E)$ we use $d(v)$ to denote its delay before upgrade and $w(v)$ to denote the cost of the upgrade.

Let $d(G)$ be the length of the longest path in G. Let $G|X$ be the graph that results when the vertices in X are upgraded. Let $d(G|X)$ be the length of the longest path in $G|X$. The *dag vertex upgrade problem* (DVUP) is that of finding a minimal weight vertex set X such that $d(G|X) \leq \delta$, where δ is an input parameter (Figure 1).



(a) G with $\delta = 6$

(b) $X = \{3, 4\}$ is the minimal solution
$d(G|X) = 6$

Figure 1: DVUP example (the first(second) coordinate is the delay(weight) of the vertex).

Throughout this paper, we assume that the weights, delays, and δ are nonnegative integers.

In [PAIK91], we showed that:

(1) DVUP is *NP*-hard for directed chains.

(2) DVUP can be solved in $O(n^3 \log n)$ time for unit weight unit delay dags (n is the number of vertices in the dag).

(3) DVUP is *NP*-hard for dags with unit weight but non unit delays when $\delta \geq 2$ and is solvable in $O(n^2)$ time when $\delta = 1$.

In addition, we proposed a backtracking algorithm and several heuristics for general dags. The related problems of vertex splitting and vertex deletion to bound path lengths were studied by Paik, Reddy, and Sahni in [PAIK90ab].

In this paper we consider the DVUP for trees, series-parallel dags (SPDAGs) and general series-parallel dags (GSPDAGs). The results we obtain are:

(1) An $O(n)$ time algorithm for unit weight unit delay trees (Section 2).

(2) Pseudo polynomial time algorithms for trees, SPDAGs and GSPDAGs with general weights and delays (Sections 2, 3, 4). These algorithms have complexity $O(n^2)$ when either $\delta = O(n)$, all weights are unit, or all delays are unit (note that if all delays are unit, then $d(G) \leq n$ and so $\delta < n$ for the problem to be nontrivial).

2. Trees

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 $d(T|X) \leq \delta$ 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 1. Note that when all vertices have unit weight, the weight of a set, Y, of vertices is simply its cardinality $|Y|$.

Theorem 1: 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 $d(T|X) \leq \delta$.

Proof: The fact that $d(T|X) \leq \delta$ 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 $d(T|X) \le \delta$. 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 $d(T|W) \le \delta$. Let Z be a minimal vertex set for which $d(T|Z) \le \delta$. 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 $T|W$ is the same as in $T|Z$. The length of the remaining paths in $T|W$ is no more than in $T|Z$. So, W is a minimal cardinality vertex set such that $d(T|W) \le \delta$ 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 $d(T(v)|X(v)) \le \delta$, $d(T(v)|W(v)) \le \delta$, and $|T(v)| \le m$ 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. General Trees

Since a chain is a special case of a tree and since DVUP for chains with arbitrary weights and delays is known to be *NP*-hard [PAIK91], 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 $d(T(v)|X) = l \le \delta$ 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 $c_1 \leq c_2$. 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 $d(T|W) \leq \delta$ has weight c'. We shall describe how to compute $S(r)$. Using the back-trace strategy of [HORO78, cf. chapter on dynamic programming] we can compute the W that results in $d(T|W) \leq \delta$ 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 given by:

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

For a 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 $(l_i, c_i) \in S(u_i)$, $1 \leq i \leq k_v$. Let $V(v)$ and $Y(v)$ be as below:

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

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

Now, $S(v)$ is the set of nondominated pairs in $V(v) \cup Y(v)$. Since $S(v)$ contains only nondominated pairs, all pairs in $S(v)$ have different l and c values. So, $|S(v)| \leq \min \{ \delta, \omega \}$ where $\omega = \sum_{u=1}^{n} w(u)$. Using the technique of [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, $\delta \leq n$ and for unit weight trees $\omega = n$. So in both of these cases the procedure described above has complexity $O(n^2)$.

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 2(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_1G_2, is obtained by identifying vertex t_1 with s_2 (Figure 2(d)). G_1G_2 is an SPDAG.

The strategy we employ for SPDAGs is a generalization of that used in Section 2.2 for trees with general delays and weights. Let s and t, respectively, be the source and sink vertices of the SPDAG G. Let $D(l,Y,G)$ be a minimum weight vertex set that contains the vertices in Y, $Y \subseteq \{s, t\}$, and such that $d(G|D(l,Y,G)) \leq l$ 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 $d(G|W) \leq \delta$ 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].

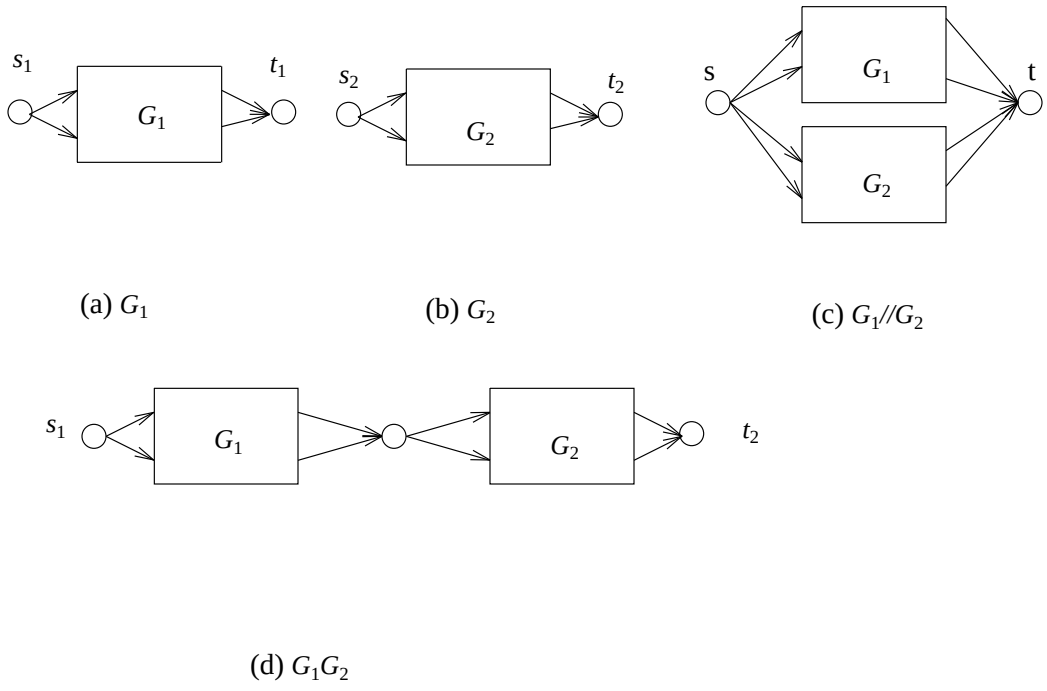


Figure 2: Series-parallel digraph.

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 $d(s) \le \delta$ **then** $F(G) := F(G) \cup \{ (d(s), w(t), \{t\}) \}$

if $d(t) \le \delta$ **then** $F(G) := F(G) \cup \{ (d(t), w(s), \{s\}) \}$

if $d(s)+d(t) \le \delta$ **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.

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

Case 1 : [$t_1 \notin D(l, Y, G_1G_2)$]

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

Case 2 : [$t_1 \in D(l,Y,G_1G_2)$]

This is similar to case 1 □

Lemma 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 $l \leq \delta$.

3.3. $G = G_1 // G_2$

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

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

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

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

Proof: Similar to that of Lemma 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 $F(G_1G_2)$. 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.

3.4. 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, $F(G_1G_2)$ 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 $O(n*\min \{\delta, \sum_{u \in V(G)} w(u)\})$. For SPDAGs with unit delay or unit weight, this is $O(n^2)$.

4. General Series Parallel Dags

General series parallel dags (GSPDAGs) were introduced in [LAWL78, MONM77, SIDN76]. A linear time algorithm to determine whether or not a given dag is a GSPDAG was developed in [VALD79]. This paper also contains a linear time algorithm to obtain a series-parallel decomposition of a GSPDAG. The definitions and terminology used in this section are derived from [VALD79].

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

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

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

Finally, a GSPDAG is a dag whose transitive reduction is an MSPDAG. The dag of Figure 4 is a GSPDAG as its transitive reduction is the MSPDAG of Figure 3(b). It is interesting to note that every dag which is a tree is an MSPDAG. However the only dag trees that are SPDAGs are chains.

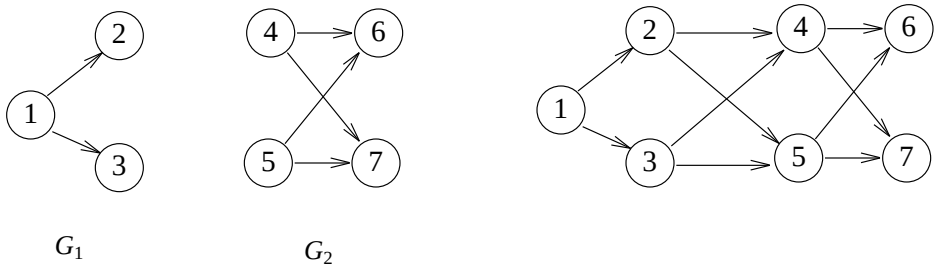


Figure 3: Parallel and series combination of G_1 and G_2.

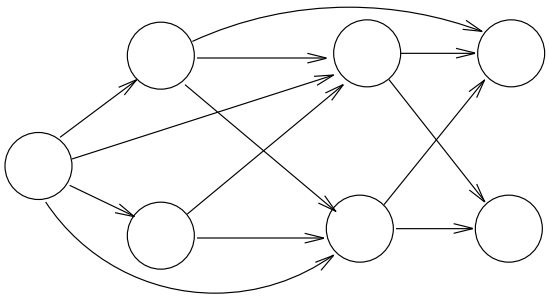


Figure 4: An example GSPDAG.

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

Lemma 3: Let $G = (V, E)$ be a GSPDAG and let $G^- = (V, E^-)$ be its transitive reduction (hence G^- is an MSPDAG). Let X be a subset of V. Then, $d(G|X) \leq \delta$ iff $d(G^-|X) \leq \delta$.

Proof: If $d(G|X) \leq \delta$ then $d(G^-|X) \leq \delta$ as $E^- \subseteq E$. If $d(G|X) > \delta$, then there is a path $P = v_1 v_2 \ldots v_q$ in G such that its delay in $G|X$ is $\Delta = \sum_{1 \leq i \leq n,\, v_i \notin X} d(v_i)$ and $\Delta > \delta$. From the v_1 to v_q path P of G we can construct a v_1 to v_q path Q in G^- by replacing each edge $< v_i, v_{i+1} > \notin E^-$ by a path from v_i to v_{i+1} in G^-. Since G^- is the transitive reduction of G, such a path exists. When all edges $< v_i, v_{i+1} >$

$\notin E^-$ have been so replaced, we obtain the desired path Q. Clearly, the delay of Q is $\geq \Delta$. Hence, if $d(G|X) > \delta$ then $d(G^-|X) > \delta$. This completes the proof. □

As a consequence of Lemma 3 the DVUP for GSPDAGs can be solved using the steps:

Step1: Compute G^-, the transitive reduction of the GSPDAG G.

Step2: Let X be the minimum weight vertex subset such that $d(G^-|X) \leq \delta$.

Step3: Output X.

Since the transitive reduction of a GSPDAG G can be obtained in time linear in the number of vertices and edges in G [VALD79], we need be concerned only with step 2. Our strategy for this is similar to that used in Section 3 for SPDAGs. However, since the source and sink vertices of G_1 and G_2 remain distinct following a series or parallel combination, we can deal with tuples (l, c) rather than with triples (l, c, Y). Corresponding to $D(l, Y, G)$ of SPDAGs, we define $D(l, G)$ for an MSPDAG G to be a minimum weight vertex set for which $d(G|X) \leq l$. We use the series parallel decomposition of G and begin with the non dominated tuple sets $F(G)$ for each of the vertices in G. Then using the series and parallel combinations that result in G we obtain $F(G)$.

4.1. G Is A Single Vertex

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

4.2. G Is Of The Form G_1G_2

The following lemma is the analogue of Lemma 1.

Lemma 4: If $(l, c) \in F(G_1G_2)$, then there is an $(l_1, c_1) \in F(G_1)$ and an $(l_2, c_2) \in F(G_2)$ such that

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

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

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

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

Proof: Similar to that of Lemma 1. □

For the case of MSPDAGs, all pairs $(l_1, c_1) \in F(G_1)$ and $(l_2, c_2) \in F(G_2)$ are compatible pairs for combination. The pair (l, c) that results from combining these two pairs is (l_1+l_2, c_1+c_2). This is added to Z (see step 1 of the procedure for G_1G_2 in Section 3.2) provided $l_1+l_2 \leq \delta$.

4.3. $G = G_1 // G_2$

The analogue of Lemma 2 that applies to MSPDAGs is given below.

Lemma 5: If $(l, c) \in F(G_1 // G_2)$, then there is an $(l_1, c_1) \in F(G_1)$ and an $(l_2, c_2) \in F(G_2)$ such that

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

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

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

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

Proof: Similar to that of Lemma 1. □

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

4.4. Complexity

While the algorithm for MSPDAGs is simpler than that for SPDAGs, its asymptotic complexity is the same. Note that for trees with unit weight and/or unit delay the complexity is $O(n^2)$ which is asymptotically the same as the algorithm of Section 2.2 for unit weight or unit delay trees but inferior to the algorithm of Section 2.1 for unit weight and unit delay trees.

5. Conclusions

We have obtained pseudo polynomial time algorithms for DVUP for trees, series-parallel dags, and general series-parallel dags. These algorithms have quadratic complexity when all vertices either have unit delay or unit weight. For $\delta = O(n)$, the complexity is $O(n\delta)$. For the case of trees with unit weights and unit delays we have developed a linear time algorithm.

6. References

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

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

[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.

[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, 1990.

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

[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.