Vertex Splitting In Dags And Applications To Partial Scan Designs And Lossy Circuits

Doowon Paik+
COmputer Science Dept
University of Minnesota
Minneapolis
MN 55455

Sudhakar Reddy++
Dept of ECE
University of Iowa
Iowa City
IA 52242

Sartaj Sahni+
CIS Dept, CSE 301
University of Florida
Gainesville
FL 32611

Communicating Author:
Sartaj Sahni
Phone: 904-392-1527
FAX: 904-392-1220

Category: 8

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

Abstract

Directed acyclic graphs (dags) are often used to model circuits. Path lengths in such dags represent circuit delays. In the vertex splitting problem, the objective is to determine a minimum number of vertices to split so that the resulting dag has no path of length δ. This problem has application to the placement of flip-flops in partial scan designs, placement of latches in pipelined circuits, placement of signal boosters in lossy circuits and networks, etc. Several simplified versions of this problem are shown to be *NP*-hard. A linear time algorithm is obtained for the case when the dag is a tree. A backtracking algorithm and heuristics are developed for general dags and experimental results using dags obtained from ISCAS benchmark circuits are obtained.

KEYWORDS and PHRASES

Partial-scan designs, flip-flop selection, sequential circuits, lossy circuits and networks, pipelined circuits, NP-hard

Vertex Splitting In Dags And Applications To Partial Scan Designs And Lossy Circuits

1 Introduction

In order to achieve high fault coverage in sequential circuits they are often designed to be easily testable. The current method of choice is the scan-design. In test mode all flip-flops in a sequential circuit, using scan-design, are connected into one or more shift registers. This allows one to set the contents of the flip-flops to the desired state as well as to observe the states of the flip-flops. As the complexity of logic circuits grows, the overhead for full scan-designs may become unacceptable. For such situations, partial-scan designs have been proposed. In partial-scan designs only a selected subset of the flip-flops in a sequential circuit are included in the scan-path. Several methods to choose the flip-flops to be included in the scan-path have been proposed [CHEN90], [GUPT90], [LEE90]. One of these proposals gives a method to use the structural information in a sequential circuit to determine the flip-flops to be placed in a scan-path [CHEN90]. We briefly discuss this method.

A sequential circuit is represented by a directed graph (digraph) called S-graph. Each flip-flop in a sequential circuit is represented by a node in the S-graph. A directed edge exists in the S-graph from node i to node j if the state of the flip-flop represented by node j depends on the state of the flip-flop represented by node i (that is ,there is a path, through combinational logic, in the circuit from the output of flip-flip i to the input of flip-flop j). Figure 1 is an example of a S-graph. Empirical evidence suggests that the existence of cycles and the maximum path length between nodes of the S-graph increase the complexity of deriving tests for sequential circuits. It was therefore suggested in [CHEN90] to include a minimum subset of flip-flops into a scan-path such that the resulting S-graph is cycle-free and the maximum distance between a pair of nodes is small. There are several cycles in the S-graph of Figure 1. If the flip-flop corresponding to node 2 is included in the scan-path then one replaces node 2 with a sink node 2^i and a source node 2^o as shown in Figure 2. This transformation corresponds to the fact that the contents of flip-flops in a scan path can be set and observed in test mode. Notice that the S-graph of Figure 2 is cycle free.

The maximum distance between node 2^o and 2^i is six. If a flip-flop corresponding to node 5 is also included in the scan-path then the S-graph of Figure 3 is obtained. In this the maximum distance between any pair of nodes is less than or equal to 3.

Two step methods to select the flip-flops to be scanned were proposed in [CHEN90], [GUPT90], and [LEE90]. In the first step a minimal subset of flip-flops is selected to be included in the scan-path such that the resulting S-graph is acyclic. In the second step additional flip-flops are selected to be included in the scan path such that in the resulting S-graph the maximum

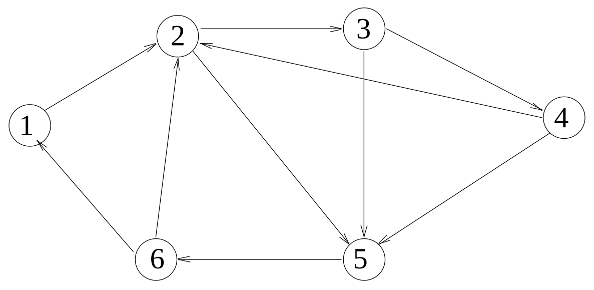

Figure 1: An example S-graph.

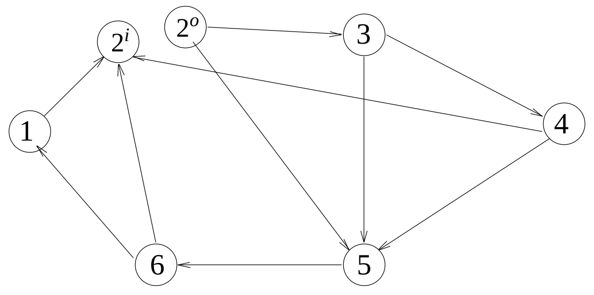

Figure 2: An acyclic S-graph for Figure 1.

distance between any pair of nodes is less than or equal to a specified number δ. This second step can be modeled as a vertx splitting problem on directed acyclic graphs (dags).

In this paper we study solutions to the problem of finding a minimum number of nodes, in a dag, to be split such that the maximum distance between any two nodes in the resulting digraph is less than or equal to a pre-specified value δ. The dags we consider are more general than the ones that arise from S-graphs. We permit each edge in the dag to have a positive integral weight instead of requiring all edges to have unit weight. This generalization can be shown to have application in the placement of latches in pipelined circuits and in the placement of signal boosters in lossy circuits.

In Section 2, we introduce the terminology we shall use in the remainder of this paper. The

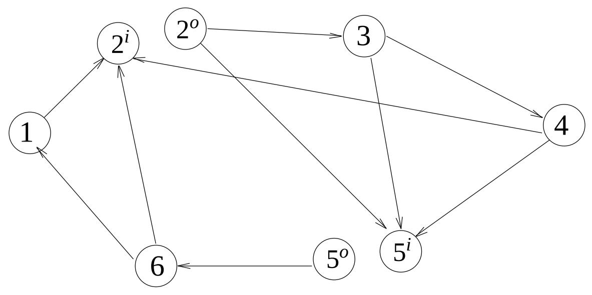

Figure 3: An S-graph with maximum distance 3.

NP-hard results are developed in Section 3 and the linear time algorithm for tree dags is given in Section 4. A backtracking algorithm and heuristics for the dag vertex splitting problem are proposed in Section 5 and 6, respectively. Section 7 reports on experiments with the ISCAS benchmark circuits. It should be noted that a linear time algorithm for series-parallel dags is easily derived from the linear time dag vertex deletion algorithm of [PAIK91].

2 Terminology

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 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 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 3 shows the result, G/X, of splitting the vertex 5 of the dag of Figure 2. 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 dag of Figure 2 and $\delta = 3$, $X = \{5\}$ is a solution to the DVSP problem.

Lemma 1: Let $G = (V,E,w)$ be a weighted dag and let δ be a prespecified delay value. Let MaxEdgeDelay = $\max_{<i,j> \in E} \{ w(i,j) \}$. Then the DVSP has a solution iff $\delta \geq$ MaxEdgeDelay.

Proof: 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 $d(G/X) \leq \delta$. □

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. DVSP for unit weight graphs with $\delta \geq 2$.
2. DVSP for unit weight multistage graphs with $\delta \geq 4$. (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. Because of space limitations, only the details of the unit weight DVSP proof are provided. The proff for unit weight multistatge graphs can be found in [PAIK90].

We shall show that the known *NP*-complete problem 3SAT can be solved in polynomial time if the unit weight DVSP with $\delta \geq 2$ can.

3SAT Problem[GARE79]

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 exactly three literals.

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

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 4(a) shows a chain with $\delta - 1$ vertices. This chain is represented by the schematic of Figure 4(b). The variable subassembly, $VS(i)$, for variable x_i is given in Figure 4(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.

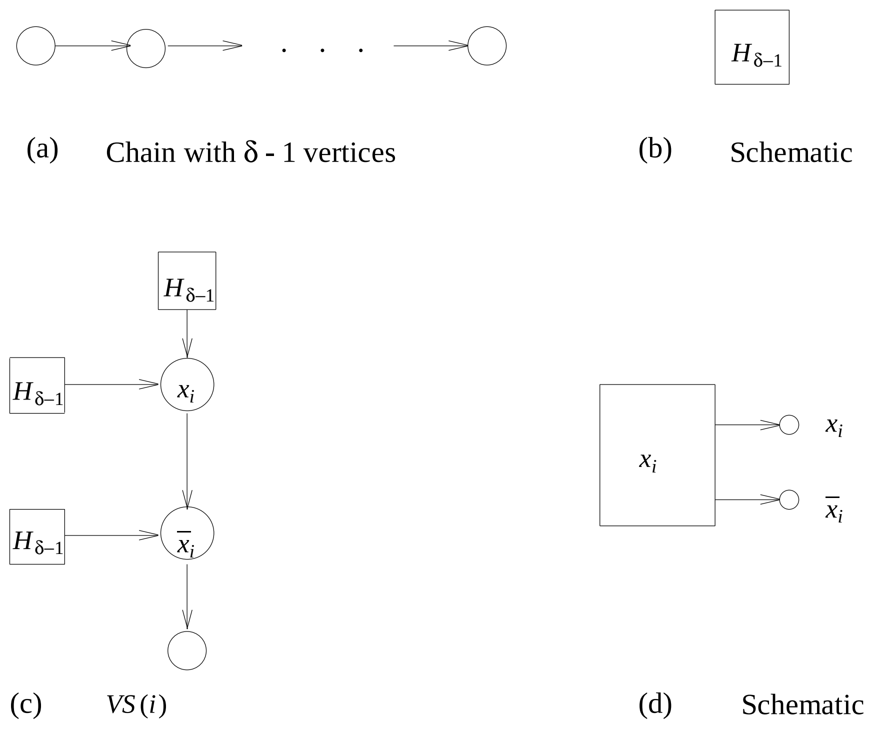

Figure 4: Variable subassembly for DVSP.

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 $X = \{ \overline{x}_i \}$. Figure 4(d) shows the schematic for $VS(i)$.

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 5(a). The schematic for $CS(j)$ is given in Figure 5(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 $F =$

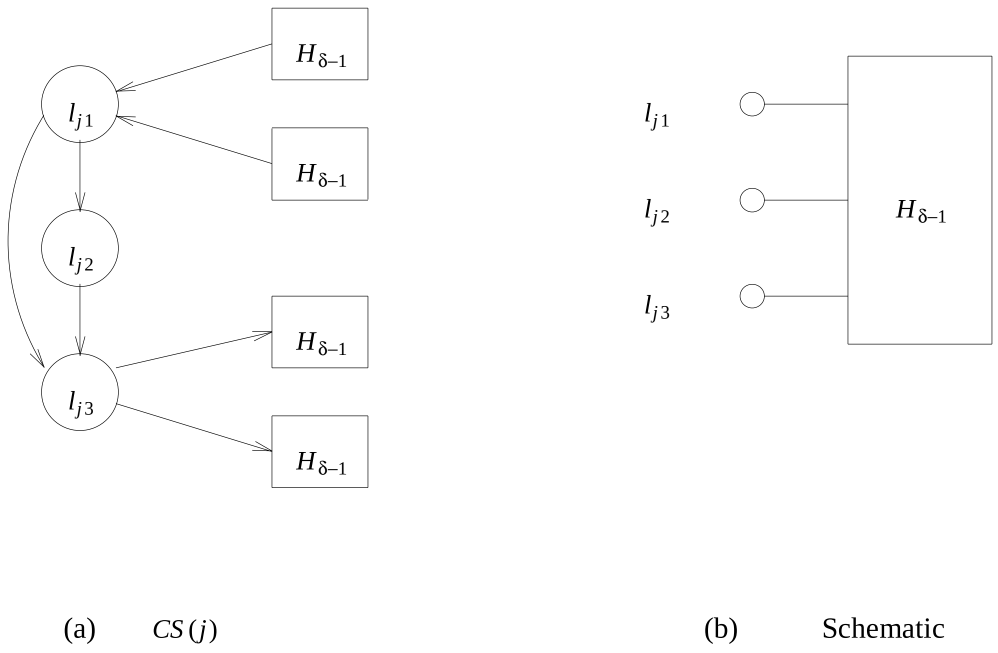

Figure 5: Clause assembly for DVSP.

$(x_1+\bar{x}_2+\bar{x}_4)\ (\bar{x}_1+\bar{x}_3+\bar{x}_4)\ (x_1+x_2+x_3)$, the G_F of Figure 6 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 1: 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 $|X| = n + 2m$.

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:

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

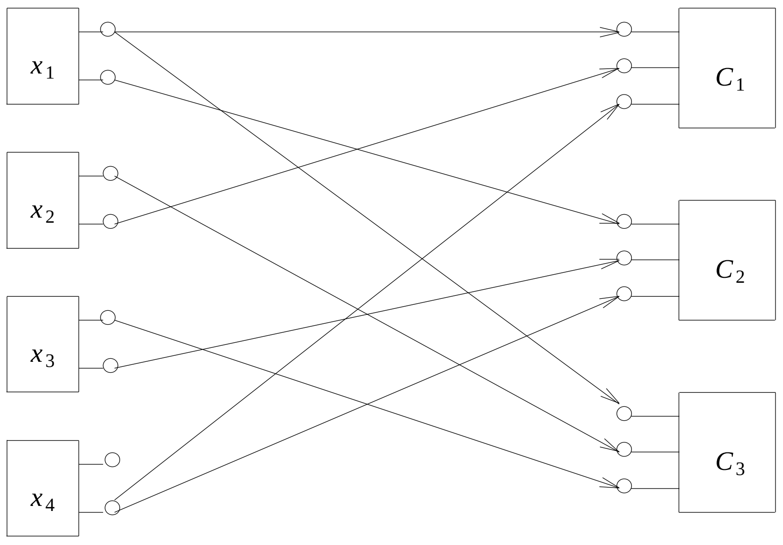

Figure 6: G_F for $F = (x_1 + \overline{x}_2 + \overline{x}_4)\ (\overline{x}_1 + \overline{x}_3 + \overline{x}_4)\ (x_1 + x_2 + x_3)$.

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 $\{x_i, \overline{x}_i\}$, $1 \le i \le n$ and exactly 2 from each of the sets $\{l_{j1}, l_{j2}, l_{j3}\}$, $1 \le j \le m$. Hence there is no i such that both $x_i \in X$ and $\overline{x}_i \in X$ 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 $l_{jk} = x_i$ $(\overline{x}_i)$. Since $d(G_F/X) \le \delta$, vertex x_i $(\overline{x}_i)$ must be split 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. □

When $\delta = 1$, the unit weight DVSP is easily solved as now every vertex that is not a source or sink has to be split.

4 Tree DVSP

In this section we develop a linear time algorithm for the DVSP 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 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. The formal algorithm is given in Figure 7. The algorithm assumes that X has been initialized to $\emptyset$ and that $w(i,j) \le \delta$ for every edge in T since otherwise, there is no solution. Its complexity is $O(n)$ where n

is the number of vertices in T.

Theorem 2: Procedure DVSP_tree finds a minimum cardinality X such that $d(T/X) \le \delta$.

Proof: See [PAIK90]. □

```
procedure DVSP_tree(T);
   {Find minimum cardinality $X$ such that $d(T/X) \le \delta$}
   {Assume that $w(i,j) \le \delta$ for every edge in $T$ and that}
   {$X$ is initialized to $\emptyset$}
begin
    if $T$ < > nil
    then begin
            $D(T) = 0$;
            for each child $Y$ of $T$ do
            begin
                DVSP_tree($Y$);
                $D(T) := \max\{D(T), D(Y)+w(T,Y)\}$;
            end;
            if $T$ is not the root
            then if $D(T) + w(parent(T),T) > \delta$
                 then begin
                        $X := X \cup \{T\}$; {split $T$}
                        $D(T) = 0$;
                      end;
         end;
end; {of DVSP_tree}
```

Figure 7: DVSP algorithm for trees.

5 A Backtracking Algorithm For DVSP

Backtracking algorithms [HORO78] generally search a tree organization of the solution space using bounding functions. The solution to our problem is a 0/1 vector $X = (x_1, x_2, \cdots, x_n)$ where n is the number of vertices and $x_i = 0$ iff vertex i is not split. We use the binary tree organization used in [HORO78] for the 0/1-knapsack problem. In this organization, the nodes at level i denote a decision on x_i, $1 \le i \le n$. If $x_i = 0$ we move to the left subtree. Otherwise we move to the right subtree of a level i node.

The remaining features of our backtracking algorithm are :

1) The vertices of the dag are considered in topological order. Thus, x_i denotes a decision on whether or not the i'th vertex, in the topological order, of the dag is split.

2) If the i'th vertex, in the topological order, is a source or sink vertex then the subtree with $x_i = 1$ is not considered as source and sink vertices are not to be split.

3) Let Y be a node in the solution space tree. If Y is at level i (root is at level 1), then the path from the root to Y determines values for $x_1, x_2, \cdots, x_{i-1}$. Let $G//Y$ be the dag obtained from the original dag by splitting the vertices with $x_j = 1$, $1 \le j < i$. Let f be the delay of the maximum delay path in $G//Y$ that ends at the i'th vertex in the topological order and let $g(G//Y)$ be the delay of the maximum delay path in $G//Y$ that begins at the i'th vertex. We use the following rules to move to a child of node Y:

   3a) If $f + w(i,j) > \delta$ for some $<i,j> \in E$, then set $x_i = 1$ and eliminate the $x_i = 0$ subtree.

   3b) If $f + g(G//Y) \le \delta$, then set $x_i = 0$ and eliminate the $x_i = 1$ subtree.

   3c) If there is only one edge $<i,j>$ that leaves vertex i, and $f + w(i,j) \le \delta$, then set $x_i = 0$ and eliminate the subtree $x_i = 1$.

   3d) If none of 3a) − 3d) apply, then search the subtree of Y with $x_i = 0$ first and later search the one with $x_i = 1$.

4) To bound a node Y, we do the following. Let opt be the number of nodes split in the best solution found so far, and let r be the number of nodes split on the path from the root to Y. And let $l(G//Y)$ be the delay of the maximum delay path in $G//Y$. It is clear that at least $\lceil l(G//Y)/\delta \rceil - 1$ additional vertices need to be split. So, if $opt \le r + \lceil l(G//Y)/\delta \rceil - 1$ then node Y is bounded and the subtree with root Y is not to be searched.

6 Heuristics For DVSP

We formulate four simple and intuitively appealing constructive heuristics to obtain a set X such that $d(G/X) \le \delta$. All four split one vertex at a time until the remaining dag has delay $\le \delta$. They assume that the input dag has a feasible solution. I.e., no edge has delay $> \delta$.

The first three heuristics have the form given in Figure 8 and differ only in the criteria used to select the next vertex to split.

6.1 Heuristic 1 (h1)

The selection criteria for the next vertex to split is :

a) $v \notin X$ and v is neither a source nor a sink vertex

b) v is on a path with delay $> \delta$

```
$X := \emptyset$ ; { $X$ is the set of vertices to split }
while $d(G/X) > \delta$ do
begin
    Select the next vertex, $v$, to split;
    $X := X \cup \{v\}$;
end;
```

Figure 8: General form of heuristics 1 through 3.

c) Of all vertices that satisfy a) and b), v has the maximum number of incident edges that are on paths of delay $> \delta$. In case of a tie, let Z be the set of vertices that are tied. For each $u \in Z$ determine $l(u)$ and $r(u)$ such that $l(u)$ is the length of a longest path from a source of G/X to u and $r(u)$ is the length of a longest path from u to a sink of G/X. The vertex with the maximum value of min $\{l(u), r(u)\}$ is selected. If there is still a tie, this is broken arbitrarily.

This heuristic is easily implemented to have run time $O(k(n+e))$ where k is the number of vertices split, n is the number of vertices in the dag, and e is the number of edges in the dag.

6.2 Heuristic 2 (h2)

In this heuristic, the next vertex, v, to split satisfies criteria a) and b) of Heuristic 1. In addition, the following criteria is employed:

c’) Of all the vertices that satisfy a) and b), v is a vertex whose splitting results in a dag that has the fewest number of vertices that are on paths of delay $> \delta$. Ties are broken as in h1.

Heuristic 2 may be implemented to have complexity $O(kne)$.

6.3 Heuristic 3 (h3)

Heuristic 3 also uses criteria a) and b) used by Heuristic 1. However, criteria c) is replaced by:

c’’) Of all the vertices that satisfy a) and b), v is such that its splitting results in a dag with least delay. I.e., v is such that $d(G/(X \cup \{v\}))$ is minimum over all choices for v. Ties are broken as in h1.

The complexity of Heuristic 3 is $O(kne)$.

6.4 Heuristic 4 (h4)

In this heuristic, the vertices of the dag are examined in two different orders: topological and reverse topological. When the *i*'th vertex in the topological (reverse topological) order is examined, it is split if the current dag contains a path comprised solely of vertices 1, ... , *i* and one additional vertex that has delay > δ. The heuristic is specified in Figure 9. It can be implemented to run in $O(n + e)$ time. Note that the additional vertex *j* can be restricted to the set of vertices adjacent to *i*.

```
X := ∅ ; { set of split vertices }
for i := 1 to n do { in topological order }
    if G/X has a path comprised solely of vertices
        1, 2, ... , i, and j (for any j) with delay > δ
    then X := X ∪ { i };

Y := ∅ ; { set of split vertices }
for i := n down to 1 do { in reverse topological order }
    if G/Y has a path comprised solely of vertices
        n, n−1, ... , i, and j (for any j) with delay > δ
    then Y := Y ∪ { i };

if |X| < |Y|
then split vertices in X
else split vertices in Y;
```

Figure 9: Heuristic 4.

7 Experimental Results

The backtracking algorithm of Section 5 and the four heuristics of Section 6 were programmed in Pascal and run on an Apollo DN3500 workstation. We experimented with two sets of acyclic directed graphs. The first set was obtained from the S-graphs of the ISCAS-89 benchmark sequential circuits [BRGL89]. The S-graphs were first rendered cycle free by the procedure given in [LEE90]. The characteristics of the resulting dags are given in Table 1. The other set of graphs was derived from the ISCAS-85 benchmark combinational circuits [BRGL85]. Here the nodes in the digraph model the gates in the circuit and the edges correspond to the connections between gates. Associated with each edge is the propagation delay along the corresponding circuit gate input. The edge delay was set to the maximum of the rising and falling delays provided in [BRGL85]. The characteristics of these circuits are given in Table 2. For each dag, G, we experimented with the δ values { $.9d(G)$, $.8d(G)$, $.7d(G)$, $.6d(G)$, $.5d(G)$, $.4d(G)$ }. Table 3 gives the results for the case G = s400. Note from Table 1 that d(s400) = 16. For δ close to d(s400) (specifically, $\delta = 12$ and 14), all four heuristics found optimal solutions. Heuristic 2 was the only one that obtained optimal solutions for all tested δ values. Table 4 gives the performance of circuit s38584. The backtracking algorithm was able to complete only for the case $\delta = .9d(G)$ and $\delta = .8d(G)$ in the time alloted for each run. Heuristic h2 consistently obtained better solutions than obtained by the remaining heuristics. However, its run time, while quite acceptable, was greater than that of heuristics 1 and 4.

Tables 5 and 6 give the total number of nodes split by each of the four heuristics for each of the sequential and combinational circuits, respectively. For each circuit the six δ values { $.9d(G)$, $.8d(G)$, $.7d(G)$, $.6d(G)$, $.5d(G)$, $.4d(G)$ } were used and the tables give the sum of the number of vertices split for each of these δ values. Table 7 and 8 give the % of tests on which each heuristic obtained the best solution. Heuristic 2, on average, was significantly better than the others.

Generally, for δ close to $d(G)$ the four heuristics tended to obtain solutions of comparable quality while for smaller δ the differences were more noticeable. However, in all δ ranges tested, heuristic 2 tended to produce the best solutions. The average run time for each of the circuits and each δ value is given in Tables 9 and 10. As can be seen heuristics 1 and 4 are very fast. While heuristic 2 is significantly faster than heuristic 3, it is much slower than h1 and h4. Despite this, we recommend h2 because it produces relatively better solutions and its run time is acceptable.

8 References

[BRGL85] F. Brglez and H. Fujiwara, "A Neutral Netlist of Ten Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. IEEE Symp. on Circuits & Systems*, June 1985 pp. 663-666.

[BRGL89] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. of Intern. Symp. on Circuit & Systems*, May 1989, pp. 1929-1934.

[CHEN90] K.T. Cheng and V. D. Agrawal, "A Partial Scan Method for Sequential Circuits with Feedback," *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 544-548, April 1990.

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

[GUPT90] R. Gupta, R. Gupta and M. A. Breuer, "BALLAST: A Methodology for Partial Scan Design," *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 538-544, April 1990.

[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.

[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.

[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.

[PAIK91] D. Paik, S. Reddy, and S. Sahni, "Deleting Verticies To Bound Path Lengths", University of Florida, Technical Report 91-04, 1991.

circuit	# vertices	# edges	$d(G)$
s400	173	282	16
s420	37	130	17
s526	27	98	12
s526n	27	98	12
s838	69	266	33
s1423	74	917	26
s5378	233	1314	20
s9234	216	1633	32
s13207	762	3083	35
s15850	608	8562	61
s35932	1777	3380	36
s38417	1396	8754	29
s38584	1448	9471	129

Table 1: Circuit characteristics (unit delay) of modified sequential circuits

circuit	# vertices	# edges	max delay
c432	250	426	57.40
c499	555	928	53.30
c880	443	729	53.00
c1355	587	1064	49.90
c1908	913	1498	76.59
c2670	1426	2076	86.87
c3540	1719	2939	98.69
c5315	2485	4386	99.30
c6288	2448	4800	319.88
c7552	3719	6144	85.30

Table 2: Circuit characteristics (with max of falling and rising delay) of ISCAS combinational circuits.

	# nodes split					run time (sec)				
$\lfloor\delta\rfloor$	h1	h2	h3	h4	optimal	h1	h2	h3	h4	optimal
14	1	1	1	1	1	< 1	< 1	< 1	< 1	< 1
12	1	1	1	1	1	< 1	< 1	< 1	< 1	< 1
11	2	2	2	3	2	< 1	< 1	< 1	< 1	< 1
9	5	4	4	7	4	< 1	< 1	1	< 1	10
8	7	4	4	11	4	< 1	< 1	1	< 1	24
6	12	8	11	10	8	< 1	1	3	< 1	35980

Table 3: Results for s400

	# nodes split					run time (sec)				
$\lfloor\delta\rfloor$	h1	h2	h3	h4	optimal	h1	h2	h3	h4	optimal
116	21	2	2	5	2	3	61	61	< 1	222
103	15	2	4	24	2	2	64	127	< 1	17280
90	18	2	5	20	-	3	68	181	< 1	-
77	27	4	6	20	-	5	127	218	< 1	-
64	40	8	13	27	-	7	293	682	< 1	-
51	89	10	37	44	-	18	439	2126	< 1	-

Table 4: Results for s38584

circuit	h1	h2	h3	h4
s400	28	20	23	33
s420	31	31	31	31
s526	21	33	25	26
s526n	21	33	25	26
s838	60	36	36	36
s1423	58	63	57	62
s5378	24	9	10	18
s9234	64	27	33	50
s13207	139	48	99	87
s15850	194	59	121	217
s35932	174	128	147	147
s38417	414	133	246	406
s38584	210	28	67	140

Table 5: Total number of nodes split for modified sequential circuits

circuit	h1	h2	h3	h4
c432	17	12	12	70
c499	48	72	108	120
c880	65	45	97	82
c1355	60	73	97	152
c1908	128	50	109	144
c2670	183	59	101	147
c3540	389	142	262	226
c5315	260	84	264	184
c6288	1413	154	298	174
c7552	564	249	709	635

Table 6: Total number of nodes split for combinational circuits

circuit	h1	h2	h3	h4
s400	50	100	83	33
s420	83	83	83	83
s526	100	33	50	50
s526n	100	33	50	50
s838	33	83	83	83
s1423	83	17	100	33
s5378	17	100	83	67
s9234	0	100	50	0
s13207	0	100	17	17
s15850	0	100	33	17
s35932	33	100	67	67
s38417	17	83	0	17
s38584	0	100	17	0
average	39.7	79.5	55.2	39.8

Table 7: Percentage of best solutions for sequential circuits

circuit	h1	h2	h3	h4
c432	33	100	100	17
c499	100	33	0	17
c880	33	83	50	0
c1355	100	17	0	33
c1908	0	100	17	0
c2670	0	100	50	33
c3540	0	100	0	0
c5315	0	100	0	17
c6288	0	83	17	17
c7552	0	100	0	17
average	26.6	81.6	23.4	15.1

Table 8: Percentage of best solutions.

circuit	h1	h2	h3	h4
s400	< 1	< 1	1	< 1
s420	< 1	< 1	< 1	< 1
s526	< 1	< 1	< 1	< 1
s526n	< 1	< 1	< 1	< 1
s838	< 1	< 1	< 1	< 1
s1423	< 1	3	3	< 1
s5378	< 1	1	1	< 1
s9234	< 1	3	5	< 1
s13207	1	37	95	< 1
s15850	5	195	379	< 1
s35932	2	532	749	< 1
s38417	11	681	2671	< 1
s38584	6	175	558	< 1

Table 9: Average run time for sequential circuits.

circuit	h1	h2	h3	h4
c432	< 1	2	2	< 1
c499	< 1	59	112	< 1
c880	< 1	14	36	< 1
c1355	< 1	70	101	< 1
c1908	1	84	194	< 1
c2670	1	168	320	< 1
c3540	6	875	2186	< 1
c5315	5	682	2607	< 1
c6288	39	3104	10235	< 1
c7552	18	6212	22632	< 1

Table 10: Average run time for combinational circuits.