Long And Short Covering Edges In Combinational Logic Circuits

Wing Ning Li+, Sudhakar M. Reddy++, Sartaj Sahni+++

ABSTRACT

This paper extends the polynomial time algorithm we obtained in [7] to find a minimal cardinality path set that long covers each lead or gate input of a digital logic circuit. The extension of this paper allows one to find, in polynomial time, a minimal cardinality path set that both long and short covers these leads or gate inputs.

KEYWORDS and PHRASES

Testing, combinational circuits.

+Department of Computer Science, University of Arkansas, Fayetteville, Arkansas 72703. Research supported by the National Science Foundation under grants DRC84-20935 and MIP86-17374.
++Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242. Research supported by Texas Instruments, Inc. and by SDIO/IST contract No. N00014-87K-0419 managed by U.S. Office of Naval Research.
+++Department of Computer Science, University of Minnesota, Minneapolis, MN 55455. Research supported by the National Science Foundation under grants DRC84-20935 and MIP86-17374.

1 INTRODUCTION

In the design of ultra-fast digital logic circuits it is important to ascertain the maximum and minimum delays suffered by signals through the circuits. The need to ascertain maximum circuit delays is quite obvious. The need to ascertain minimum circuit delays arises due to requiremnts on data hold times at the inputs to flip-flops, the data skew and other timing constraints in high speed pipelines [19], insuring correct data at the inputs of edge triggered flip-flops, and in the design of reliable asynchronous sequential logic circuits [17].

In order to verify that the delays along all circuit paths are within specified upper and lower bounds one can attempt to test all circuit paths. Unfortunately such an approach would be impractical due to the large number of paths in a circuit. A more practical approach is to test enough paths such that each circuit lead is included in at least one path. The set of paths that are to be tested should be such that a "robust" path delay fault detecting test exists for each path in the test [20]. Methods to design combinational logic circuits such that every path in the circuit has a robust path delay fault detecting test have been proposed [18,21]. In such testable circuits one can select a set of circuit paths such that each circuit lead is included in at least one path. The model we use is directly applicable to these circuits. In other circuits, it is possible for several paths to be not testable. In this case it will be necessary to iterate on our algorithm until a set of testable paths has been obtained.

Verification of signal propagation in logic circuits is essential to ensure correct operation. Such verification is necessary to determine reliable speed of operation and usable clock frequencies. To perform such verification one normally choses a collection of paths to "test" [1-2,4-

5,7,9,12-15]. Of the several methods proposed [9,12] to select paths to be tested, one is to select a set, MaxSP, of paths such that for each lead l in the given circuit, there is at least one input to output path in MaxSP which exhibits maximum modeled delay among all circuit paths that contain l. We say that MaxSP *long covers* the leads of the circuit. Li, Reddy, and Sahni [7] have developed a polynomial time algorithm to find a minimum cardinality MaxSP.

A set of paths MinSP such that for each lead l there is at least one input to output path in MinSP which exhibits minimum modeled delay among all circuit paths that contain l is also useful in verifying correct circuit operation. MinSP *short covers* the leads of the circuit. The algorithm of [7] is easily modified to find a minimum cardinality MinSP.

When testing under minimum and maximum propagation delays, one really needs a set, MinMaxSP, of input to output paths such that for each lead l there is at least one path in MinMaxSP which exhibits minimum modeled delay and at least one (not necessarily different) which exhibits maximum modeled delay. The need to test the shortest propagation delays through circuit paths occurs in the design of asynchronous sequential logic circuits[11], synchronous sequential logic circuits with data driven clocks, and in the design of sequential circuits using a single (instead of multiple or two-phase non-overlapping) clock signal. MinMaxSP both *short* and *long covers* the leads in the circuit. It is easy to see that if X is a MinSP set and Y a MaxSP set, then $X \cup Y$ is a MinMaxSP set. However, one can easily construct circuits with the property that $|Z| = (|X| + |Y|)/2$ where X, Y, and Z are, respectively, minimal cardinality MinSP, MaxSP and MinMaxSP sets. This, for example, is the case when all input to output paths have the same length and the circuit contains two disjoint path sets A and B which are of minimum cardinality and which include all circuit leads. In this case A and B are, respectively, minimum cardinality MinSP and MaxSP sets. Also, either A or B could be used as a minimum cardinality MinMaxSP set.

In this paper we show how to find, in polynomial time, a minimum cardinality set MinMaxSP for a given combinational logic circuit. Combinational circuit verification is used to verify the sequential circuit delays. Since our algorithm is very closely related to that of [7], we briefly review this algorithm in Section 2. The algorithm to find a minimum cardinality MinMaxSP is developed in Section 3, and experimental results are provided in Section 4.

2 TERMINOLOGY AND REVIEW OF [7]

Li, Reddy, and Sahni [7] have shown how a combinational logic circuit with possibly different gate propagation delays for rising and falling transitions can be modeled by a network N which is a directed acyclic graph in which each edge has a single delay associated with it (Figure 1). Every vertex in N with indegree 0 is a *source* vertex. A *sink* vertex is one which has outdegree 0. Vertices 1 and 2 are the source vertices of Figure 1 while 7 and 8 are its sink vertices. The weight of a source to sink path L is the sum of the weights of the edges in L. The path L *long covers* the edge $<i,j>$ iff:

i) $<i,j>$ is an edge of L

ii) the weight of L is maximum amongst all paths L that contain edge $<i,j>$.

A set, X, of source to sink paths is a *long cover* of the network iff every edge of N is *long covered* by at least one path in X. The path set $X = \{13468, 1357, 2457, 2468\}$ is a minimum car-

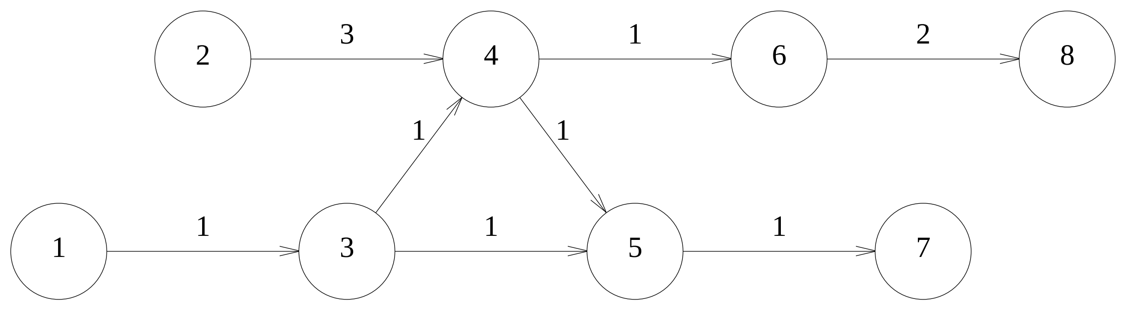


Figure 1: An example network

dinality *long cover* of the network of Figure 1.

The network model, N, of a circuit, C, obtained in [7] has the property that from a minimum cardinality *long cover* of N one easily obtains a minimum cardinality MaxSP for C. The network model N is transformed into a directed acyclic graph (dag) G_L. A source to sink path in G_L covers an edge $<i,j>$ iff $<i,j>$ is on the path. A set of source to sink paths is a *dag cover* iff each edge $<i,j>$ of G_L is on at least one path in the set. The G_L obtained from N has the property that from any minimum cardinality *dag cover* of G_L one can easily obtain a minimum cardinality *long cover* of N and in turn a minimum cardinality MaxSP of the modeled circuit C.

To obtain G_L from N, the edges of N are first classified into one of the categories Lyy, Lny, Lyn, and Lnn. Let $source(i)$ denote the set of paths in N that begin at a source vertex of N and end at vertex i. Let $sink(i)$ denote the set of paths in N that begin at i and end at a sink vertex of N. Let $longest(X)$ be the set of longest paths in X. The classification for an edge $<i,j>$ is defined as:

1) type Lyy —— $<i,j>$ is on a path in $longest(sink(i))$ and $longest(source(j))$

2) type Lny —— $<i,j>$ is not on any path in $longest(sink(i))$ but is on a path in

$longest(source(j))$

3) type *Lyn* —— $<i,j>$ is on a path in $longest(sink(i))$ but not on any path in $longest(source(j))$

4) type *Lnn* —— $<i,j>$ is not on any path in $longest(sink(i))$ or $longest(source(j))$.

G_L is now obtained from N by replacing each edge $<i,j>$ of type *Lnn*, *Lyn*, or *Lny* by a new edge as given in the table of Figure 2. This replacement introduces new vertices as indicated.

edge type	new edge	new vertex
Lnn	$<l_{ij},r_{ij}>$	l_{ij},r_{ij}
Lyn	$<i,r_{ij}>$	r_{ij}
Lny	$<l_{ij},j>$	l_{ij}

Figure 2: Replacement for edge $<i,j>$

A minimum cardinality *dag cover* for G_L is obtained by modeling G_L as a flow network and obtaining a minimum flow.

3 OBTAINING A MINIMUM CARDINALITY MinMaxSP

3.1 MINIMUM CARDINALITY MinSP

First, consider the problem of obtaining a minimum cardinality path set to short cover the circuit leads. As in [7], the circuit is modeled by a network N. This modeling is identical to that for long covering. A source to sink path L in N *short covers* the edge $<i,j>$ iff:

i) $<i,j>$ is an edge of L

ii) the weight of L is minimum amongst all source to sink paths that contain edge $<i,j>$.

A *short cover* of a network N is a set of source to sink paths such that each edge of N is short covered by at least one path in the path set. The path set {13457,13468,1357,2457} is a minimum cardinality short cover of the network of Figure 1. From a minimum cardinality *short cover* of N, one can obtain a minimum cardinality *short cover* of the modeled circuit C in the same way as one obtains a *long cover* of C from a *long cover* of N [7].

To obtain a minimum cardinality short cover of N, one constructs a dag G_S in a manner similar to the construction of G_L. Let $shortest(X)$ denote the set of shortest paths in X. The edges in N are classified as below:

5) type Syy —— $<i,j>$ is on a path in $shortest(sink(i))$ and $shortest(source(j))$

6) type Sny —— $<i,j>$ is not on any path in $shortest(sink(i))$ but is on a path in $shortest(source(j))$

7) type Syn —— $<i,j>$ is on a path in $shortest(sink(i))$ but not on any path in $shortest(source(j))$

8) type Snn —— $<i,j>$ is not on any path in $shortest(sink(i))$ or $shortest(source(j))$.

G_S is obtained from N by replacing each edge $<i,j>$ of type Snn, Syn, or Sny by a new edge as given in the table of Figure 3. The proof of [7] is easily modified to show that there is a one-to-one correspondence between *dag covers* for G_S and *short covers* for N. Further, from a minimum cardinality *dag cover* for G_S a corresponding minimum cardinality *short cover* for N is obtained in the same manner as for *long covers*. A minimum cardinality *dag cover* for G_S is obtained using a network flow model identical to that for G_L.

edge type	new edge	new vertex
Snn	$<l_{ij}, r_{ij}>$	l_{ij}, r_{ij}
Syn	$<i, r_{ij}>$	r_{ij}
Sny	$<l_{ij}, j>$	l_{ij}

Figure 3: Replacement for edge $<i,j>$

Our algorithm to find a minimum cardinality MinMaxSP begins with the dags G_L and G_S and constructs a new dag G_{LS} with the property that a minimum cardinality *dag cover* for G_{LS} corresponds to a minimum cardinality cover for N that both long and short covers the edges of N. This in turn corresponds to a minimum cardinality MinMaxSP of the modeled circuit. Since a minimum cardinality long and short cover of N may contain a path that long covers some edges and short covers others, we need to understand the conditions under which this may occur. For this, we study some properties of the paths and edges in N.

3.2 PATH AND EDGE PROPERTIES

From our earlier discussion, we know that each edge has an *Luv* and *Swx*, $u,v,w,x \in \{y,n\}$ classification. In addition to these, we provide an edge $<i,j>$ with a third classification:

$G1$: $longest(source(i)) = shortest(source(i))$

and $longest(sink(j)) = shortest(sink(j))$.

$G2$: $longest(source(i)) = shortest(source(i))$

and $longest(sink(j)) \neq shortest(sink(j))$.

$G3$: $longest(source(i)) \neq shortest(source(i))$

and $longest(sink(j)) = shortest(sink(j))$.

$G4$: $longest(source(i)) \neq shortest(source(i))$

and $longest(sink(j)) \neq shortest(sink(j))$.

Notice that $longest(source(i)) = shortest(source(i))$ iff all paths from a source vertex to vertex i have the same length; $longest(source(i)) \neq shortest(source(i))$ iff at least two source to i paths have different lengths; $longest(sink(j)) = shortest(sink(j))$ iff all paths from j to a sink vertex are of the same length; and $longest(sink(j)) \neq shortest(sink(j))$ iff at least two paths from j to a sink have different lengths.

Lemma 1: Let P be a path in N. If P contains an edge $<i,j>$ of type $G2$, then all edges preceding $<i,j>$ are of type $G2$.

Proof: If $<k,l>$ precedes $<i,j>$ and $longest(source(k)) \neq shortest(source(k))$, then there are at least two paths of different length from source vertices to k and hence to i (as there is a path from k to i in P). This contradicts the requirement on vertex i that $shortest(source(i)) = longest(source(i))$. So, $longest(source(k)) = shortest(source(k))$. Since $<i,j>$ is of type G2, $longest(sink(j)) \neq shortest(sink(j))$ and since there is a path from to l to j, $longest(sink(l)) \neq shortest(sink(l))$. So, $<k,l>$ is a G2 edge. □

Lemma 2: Let P be a path in N. If P contains an edge $<i,j>$ of type $G3$, then all edges following

$<i,j>$ are of type $G3$.

Proof: Similar to that of Lemma 1. □

Lemma 3: No path P in N can contain both a G1 and a G4 edge.

Proof: Suppose there is a path P that contains a G1 edge $<i,j>$ that precedes a G4 edge $<k,l>$. Since $longest(sink(l)) \neq shortest(sink(l))$, there are at least two paths of different lengths from l to sinks. Hence, there are at least two paths of different lengths from j to sinks. So, $longest(sink(j)) \neq shortest(sink(j))$. But, since $<i,j>$ is a G1 edge, $longest(sink(j)) = shortest(sink(j))$. A contradiction. If $<i,j>$ follows $<k,l>$ a contradiction is similarly obtained. Hence, there is no path that contains both a G1 and a G4 edge. □

Lemma 4: Every source to sink path in N that includes an edge $<i,j>$ of type $G1$ both long and short covers $<i,j>$.

Proof: Since $longest(source(i)) = shortest(source(i))$ and $longest(sink(j)) = shortest(sink(j))$, all source to sink paths that include $<i,j>$ are of the same length. Hence $<i,j>$ is both long and short covered by each such path. □

Lemma 5: Let $<i,j>$ be of type $G2$ and let P be a path that includes $<i,j>$.

a) If P long covers $<i,j>$, then neither $<i,j>$ nor any of the edges that precede it on the path P are short covered by P.

b) If P short covers $<i,j>$, then neither $<i,j>$ nor any of the edges that precede it on the path P are long covered by P.

Proof: a) Since P long covers $<i,j>$, the segment Y of P that follows the edge $<i,j>$ must be in $longest(sink(j))$. Since $<i,j>$ is of type G2, $longest(sink(j)) \neq shortest(sink(j))$. So, Y is not in $shortest(sink(j))$. Hence, P cannot short cover $<i,j>$ or any of the edges that precede it on path P. The proof for b) is similar. □

Corollary 1: No path can short cover one $G2$ edge and long cover another (possibly the same) $G2$ edge.

Proof: Suppose that some path P short covers some G2 edge $<i,j>$. Then from Lemma 5 b), it follows that P cannot long cover $<i,j>$ or any of the edges that precede it on P. If P long covers some G2 edge $<k,l>$ that follows $<i,j>$, then from Lemma 5 a) it follows that P cannot short cover $<i,j>$ (as $<i,j>$ precedes $<k,l>$). This contradicts the assumption that P short covers $<i,j>$. So, P cannot long cover any $G2$ edge. The proof for the case when P long covers some G2 edge is similar. □

Lemma 6: Let $<i,j>$ be of type $G3$ and let P be a path that includes $<i,j>$.

a) If P long covers $<i,j>$, then neither $<i,j>$ nor any of the edges that follow it on the path P are short covered by P.

b) If P short covers $<i,j>$, then neither $<i,j>$ nor any of the edges that follow it on the path P are long covered by P.

Proof: a) Since $<i,j>$ is of type G3, $longest(source(i)) \neq shortest(source(i))$. Consequently, the segment Y of P that precedes the edge $<i,j>$ is not in $shortest(i)$. Hence, P cannot short cover $<i,j>$ or any of the edges that follow it. The proof for b) is similar. □

Corollary 2: No path can short cover one $G3$ edge and long cover another (possibly the same) $G3$ edge.

Proof: Suppose that some path P short covers the G3 edge $<i,j>$. From Lemma 6 b), it follows that P cannot long cover $<i,j>$ or any of the edges that follow it. If P long covers some G3 edge $<k,l>$ that precedes $<i,j>$, then form Lemma 6 a) it follows that P cannot short cover any of the edges that follow $<k,l>$. In particular, P cannot short cover the edge $<i,j>$. This contradicts the assumption on P. Hence, P cannot long cover any $G3$ edge. In a similar manner, we can show that a path that long covers a G3 edge cannot short cover a G3 edge. □

Lemma 7: Let $<i,j>$ be of type $G4$ and let P be a path that includes $<i,j>$.

a) If P long covers $<i,j>$, then it short covers no edge in P.

b) If P short covers $<i,j>$, then it long covers no edge in P.

Proof: a) Since $<i,j>$ is a G4 edge, $longest(source(i)) \neq shortest(source(i))$ and $longest(sink(j)) \neq shortest(sink(j))$. Since P long covers $<i,j>$, the segment Y of P that follows $<i,j>$ is in $longest(sink(j))$ and the segment Z of P that precedes $<i,j>$ is in $longest(source(i))$. Consequently, Y is not in $shortest(sink(j))$ and Z is not in $shortest(source(i))$. Hence, P cannot short cover any edge. The proof for b) is similar. □

Lemma 8: A source to sink path P long covers a $G2$ edge $<i,j>$ and short covers a $G3$ edge $<k,l>$ iff:

i) $<k,l>$ is a successor of $<i,j>$ in P

ii) The path segment of P from the source vertex to k is in $shortest(source(k))$

iii) The path segment of P from j to the sink vertex is in $longest(sink(j))$.

Proof: First consider the "only if" part. Assume that P long covers the $G2$ edge $<i,j>$ and short covers the $G3$ edge $<k,l>$. From Lemma 5, it follows that $<k,l>$ must be a successor of $<i,j>$. For ii) and iii), we note that P has the form $P_1<i,j>P_2<k,l>P_3$. Since P short covers $<k,l>$, $P_1<i,j>P_2 \in shortest(source(k))$. Also, since P long covers $<i,j>$, $P_2<k,l>P_3 \in longest(sink(j))$.

Next, consider the "if" part. We may assume that both $<i,j>$ and $<k,l>$ are on P. We need to show that conditions i) - iii) imply that $<i,j>$ is long covered and $<k,l>$ is short covered. Since $<k,l>$ is to the right of $<i,j>$, P is of the form $P_1<i,j>P_2<k,l>P_3$. Let $P_L<i,j>P_R$ be some path in N that long covers $<i,j>$. We need to show that P has the same length as this path. Since, $P_2<k,l>P_3 \in longest(sink(j))$, and $P_L<i,j>P_R$ long covers $<i,j>$, $P_2<k,l>P_3$ and P_L have the same length. Also, since $<i,j>$ is of type $G2$, P_1 and P_L have the same length. So, P long covers $<i,j>$.

Let $P_L<k,l>P_R$ be some path in N that short covers $<k,l>$. P short covers $<k,l>$ iff its length is the same as that of $P_L<k,l>P_R$. The lengths of P_L and $P_1<i,j>P_2$ are the same as both are in $shortest(source(k))$. Since $<k,l>$ is of type $G3$, $longest(sink(l)) = shortest(sink(l))$. So P_3 and P_R have the same length. Hence, P and $P_L<k,l>P_R$ are of the same length.□

Lemma 9: If a source to sink path P long covers a $G2$ edge $<i,j>$ and short covers a $G3$ edge $<k,l>$, then all paths between j and k have the same length.

Proof: From Lemma 8 i) it follows that $<k,l>$ is a successor of $<i,j>$ on P. So, there is at least one path from j to k in the network. Let P be of the form XYZ where X is the segment of P from source to vertex j (i.e., the last edge in X is $<i,j>$), Y is the segment from vertex j to veretex k, and Z is the segment that begins with edge $<k,l>$ and ends at the sink vertex. From Lemma 8 ii) and iii), it follows that XY is in $shortest(source(k))$ and YZ is in $longest(sink(j))$. Now, suppose that the newtork has a path W from j to k whose length is different from that of the segment Y. If its length is less than the length of Y, then XW is a shorter source to k path than XY and so XY cannot be in $shortest(source(k))$. A contradiction. On the other hand, if the length of W is more than that of Y, then WZ is a longer j to sink path than YZ and so YZ cannot be in $longest(sink(j))$. A contradiction. So, there is no j to k path W whose length is different from that of Y. Hence, all j to k paths in the network are of the same length. □

Lemma 10: A source to sink path P short covers a $G2$ edge $<i,j>$ and long covers a $G3$ edge $<k,l>$ iff:

i) $<k,l>$ is a successor of $<i,j>$ in P

ii) The path segment of P from the source vertex to k is in $longest(source(k))$

iii) The path segment of P from j to the sink vertex is in $shortest(sink(j))$.

Proof: Similar to that of Lemma 8. □

Lemma 11: Let P be a path that short covers a non $G1$ edge $<i,j>$ and long covers a non $G1$ edge $<k,l>$. P has the form $P_L P_M P_R$ where P_L consists solely of $G2$ edges, P_R consists solely of $G3$ edges and P_M is either empty or consists solely of $G1$ edges or solely of $G4$ edges. Neither P_L nor P_R is empty.

Proof: If $<i,j>$ is a G4 edge, then from Lemma 7 it follows that P cannot long cover any edge. However, by assumption, P long covers $<k,l>$. So, $<i,j>$ is not a G4 edge. Similarly, $<k,l>$ is not a G4 edge. If $<i,j>$ is a G2 edge, then from Corollary 1 $<k,l>$ cannot be a G2 edge and so must be a G3 edge. If $<i,j>$ is a G3 edge, then from Corollary 2 $<k,l>$ cannot be a G3 edge and so must be a G2 edge. Hence, there are two cases to consider:

a) $<i,j>$ is a G2 edge and $<k,l>$ is a G3 edge

b) $<i,j>$ is a G3 edge and $<k,l>$ is a G2 edge.

Let us consider case a) first. Let $<i',j'>$ be the last G2 edge on P and let $<k',l'>$ be the first G3 edge on P. From Lemma 1, it follows that all edges that precede $<i',j'>$ in P are G2 edges and from Lemma 2, it follows that all edges that follow $<k',l'>$ in P are G3 edges. So, $<i',j'>$ precedes $<k',l'>$. Let P_L be the segment of P from the source vertex up to and including $<i',j'>$ and let P_R be the be the segment of P from (and including) $<k',l'>$ to the end of P. Let P_M be the segment between P_L and P_R. We have already shown that P_L consists solely of G2 edges, P_R consists solely of $G3$ edges, P_L includes at least the edge $<i,j>$ and so is not empty, and P_R includes at least the edge $<k,l>$ and so is not empty. It remains to show that P_M consists solely of G1 edges or solely of G4 edges. From our selection of $<i',j'>$ and $<k',l'>$ it follows that P_M cannot contain any G2 or G3 edges. Also, from Lemma 3, P_M cannot contain both a G1 and a G4 edge. So, if P_M is not empty, then it consists solely of G1 or solely of G4 edges.

Now, we prove the Lemma for case b). Let $<i',j'>$ be the first G3 edge on P and let $<k',l'>$ be the last G2 edge on P. From Lemma 1, it follows that all edges that precede $<k',l'>$ in P are G2 edges and from Lemma 2, it follows that all edges that follow $<i',j'>$ in P are G3 edges. So, $<k',l'>$ precedes $<i',j'>$. Let P_L be the segment of P from the source vertex up to and including $<k',l'>$ and let P_R be the be the segment of P from (and including) $<i',j'>$ to the end of P. Let P_M be the segment between P_L and P_R. P_L consists solely of G2 edges, P_R consists solely of $G3$

edges, and neither P_L nor P_R is empty. From our selection of $<i',j'>$ and $<k',l'>$ it follows that P_M cannot contain any G2 or G3 edges. Also, from Lemma 3, P_M cannot contain both a G1 and a G4 edge. So, if P_M is not empty, then it consists solely of G1 or solely of G4 edges. □

Lemma 12: Let P be as in Lemma 11.

a) If $<i,j>$ is in P_L, then the last short covered edge in P is of type *Syn* or *Snn* and the first long covered edge is of type *Lny* or *Lnn*.

b) If $<k,l>$ is in P_L, then the last long covered edge in P is of type *Lyn* or *Lnn* and the first short covered edge is of type *Sny* or *Snn*.

Proof: We prove only a). The proof for b) is similar. By assumption, $<i,j>$ is a short covered edge and it is in P_L. So, we are in case a) of the proof of Lemma 11. Hence, $<i,j>$ is a $G2$ edge and $<k,l>$ is a $G3$ edge which must be part of P_R. Let $<a,b>$ be the last short covered edge of P. From Corollary 2, it follows that this edge cannot be a G3 edge. Hence, it must be part of P_L or P_M. In either case, it precedes P_R. Let $<b,c>$ immediately follow $<a,b>$ in P ($<b,c>$ exists as P_R is not empty). We need to show that $<a,b>$ is not on any path in $shortest(source(b))$. Suppose $<a,b>$ is on a shortest path Q from some source vertex to b. Q has the form $X<a,b>$ and P has the form $P'<a,b><b,c>P''$. Since $<a,b>$ is short covered by P, X and P' have the same length. Since P short covers $<a,b>$, $<b,c>P''$ is a shortest path from b to a sink. Hence, $X<a,b><b,c>P''$ short covers $<b,c>$. So, $P=P'<a,b><b,c>P''$ short covers $<b,c>$. This contradicts the assumption that $<a,b>$ is the last short covered edge. Consequently, $<a,b>$ is not on any path in $shortest(source(b))$. Hence $<a,b>$ is of type *Syn* or *Snn*.

Since $<i,j>$ is short covered and of type $G2$, Corrolary 1 implies that the long covered edges cannot be of type G2. From Lemma 11, it follows that the long covered edges must be in P_M and/or P_R and so must follow P_L. Let $<f,g>$ be the first long covered edge. We need to show that $<f,g>$ is not on any path in $longest(sink(f))$. Let $<e,f>$ be the edge that immediately precedes it on P. Such an edge must exist as by Lemma 11 P_L is not empty. If $<f,g>Y$ is in $longest(sink(f))$, then the length of Y must equal that of P'' where $P = P'<e,f><f,g>P''$ as $<f,g>$ is long covered by P. Hence, $<f,g>P'' \epsilon\, longest(sink(f))$. Since $<f,g>$ is long covered by P, P' is a longest path from a source to e. Now, since $P' \epsilon\, longest(source(e))$ and $<f,g>P'' \epsilon\, longest(sink(f))$, P must long cover $<e,f>$. This contradicts the assumption on $<f,g>$. So, $<f,g>$ is on no path in $longest(sink(f))$ and therefore must be of type Lny or Lnn. □

3.3 CONSTRUCTION OF G_{LS}

The network G_{LS} is to have the property that a minimum cardinality cover (by paths) of its edges corresponds to a minimum cardinality MinMaxSP of the original network. Let H be the graph $G_L \cup G_S$. That is, the vertices in H are the vertices in G_L and G_S and the edges in H are those in G_L as well as those in G_S. Since the vertices in G_L and G_S have the same labels, it is necessary to relabel these in H. The relabeling scheme we use prefixes each vertex label in G_L with an l and each vertex label in G_S with an s. Figures 4 and 5, respectively, give the G_L and G_S networks that correspond to the network of Figure 1. The vertices have been relabeled as stated. The two figures together define H. A source to sink path in H corresponds to a source to sink path in the network N of Figure 1. If the H path is in the G_L (G_S) part of H, then the corresponding path in N

is obtained by first mapping the *H* edges back to the *N* edges and then extending the resulting path of *N* to a sink and source using a longest (shortest) such extension; the path of *N* so obtained long (short) covers the edges on the path. So, at present we only have the capability to generate paths that either long cover or short cover edges. To allow for a path to simultaneously long cover and short cover edges we need to modify *H* so that paths from the G_L component can cross into the G_S component and vice versa.

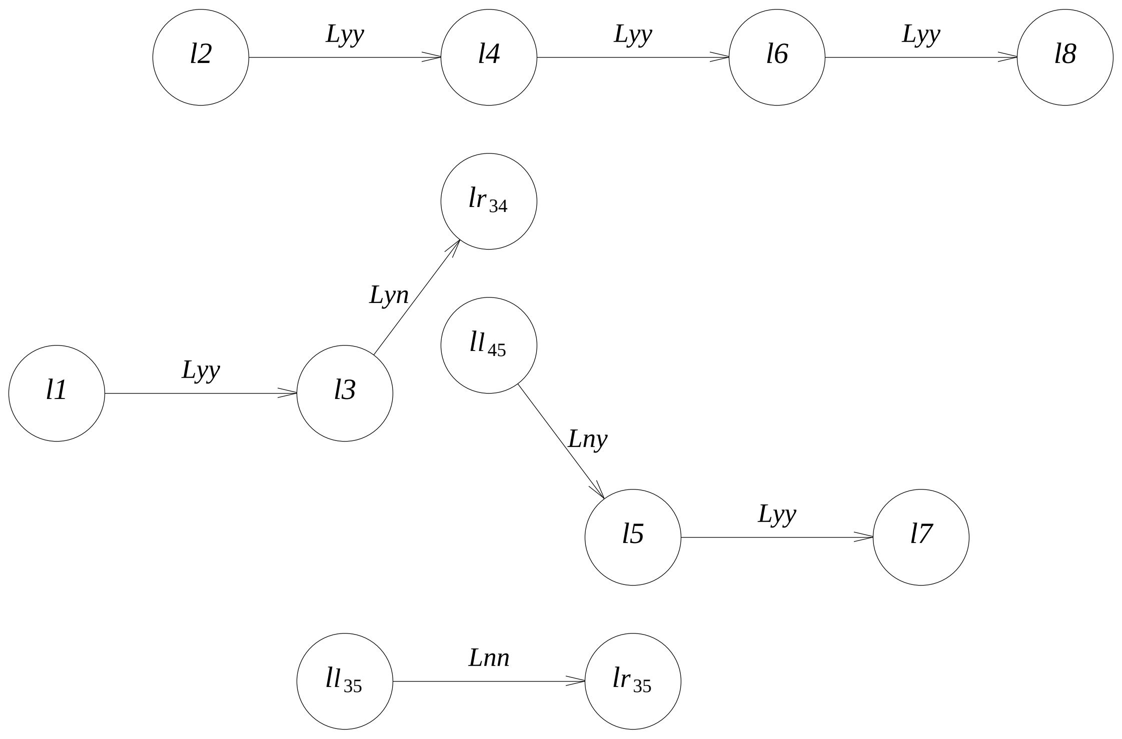


Figure 4: G_L obtained from network in Figure 1.

From Lemma 4, we see that G1 edges are long and short covered by all paths. So, we can modify *H* so as not to require two separate paths (one that long covers the G1 edge and another that short covers it). This is accomplished using the transformation of Figure 6. In this figure

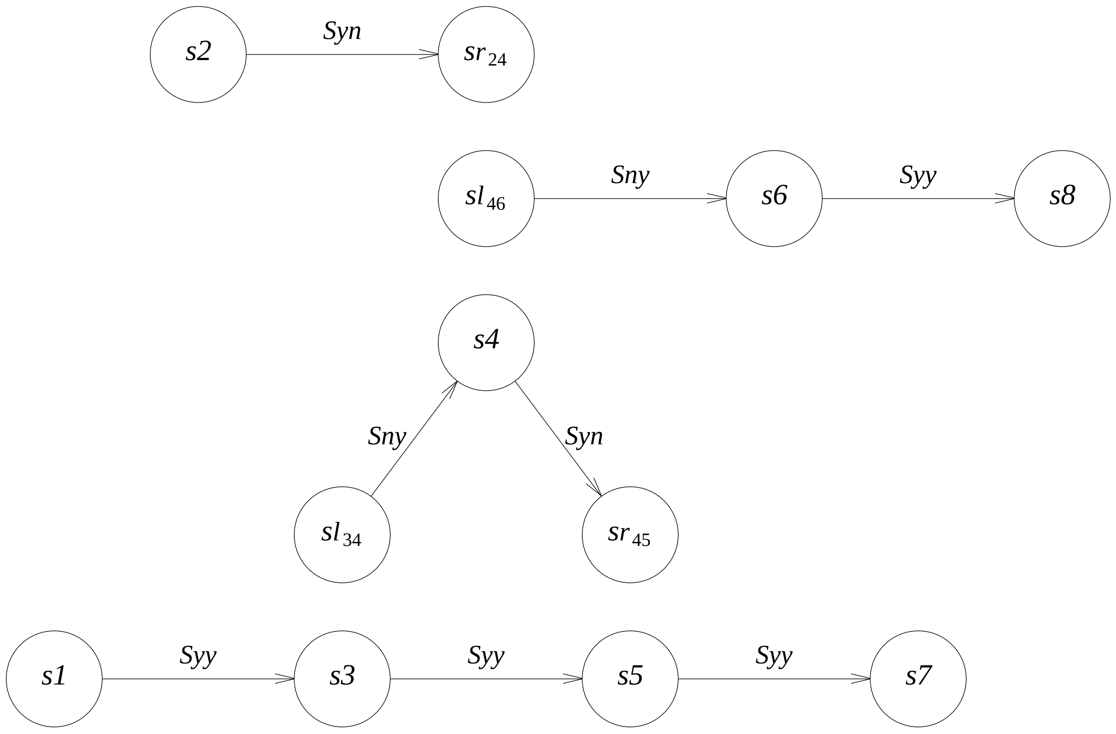


Figure 5: G_S obtained from network in Figure 1.

$<a,b>$ and $<c,d>$ are, respectively, the images of the same G1 edge $<i,j>$ in G_L and G_S. y and z are two new vertices. When covering the edges of the resulting network H' we relax the covering requirement so that edges of the type $e1$ through $e4$ (Figure 6) need not be on any path in the cover. However, all edges of type $e5$ must be on at least one path in the cover. We refer to this relaxed notion of cover as partial cover and define it more precisely later. $e5$ is now the image of the G1 edge $<i,j>$. Since the resulting network has only one image for each G1 edge and since edges of type $e1$ through $e4$ are not required to be on a path of a partial cover, the transformation of Figure 6 makes it possible to cover the image of each G1 edge by a single path in the (partial) cover. Without this transformation, each G1 edge would have two images and each image would have to be on at least one path in the cover.

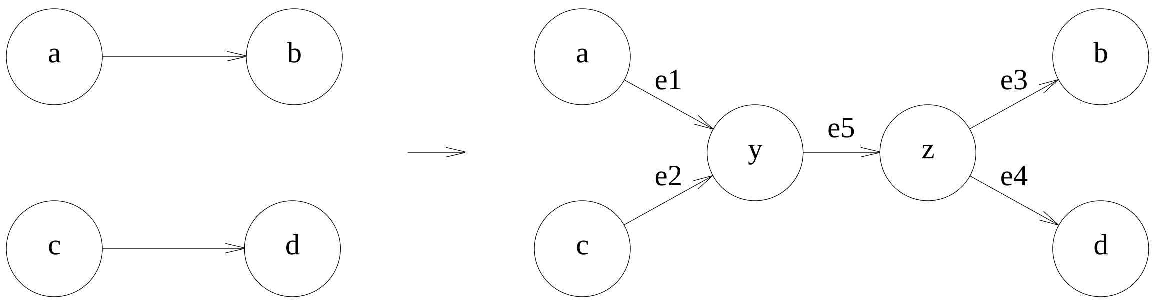


Figure 6: Step 2 transformation for $G1$ edges

From Lemma 7, paths that long (short) cover a G4 edge cannot short (long) cover any edge. So, for G4 edges no path cross overs between the G_L and G_S copmponents of H are to be provided. We do, however, need to provide for paths of the type described by Lemmas 8 through 12. For this we need to provide path connections from G2 edges of type *Syn* and *Snn* to G3 edges of type *Lny* and *Lnn* as well as from G2 edges of type *Lyn* and *Lnn* to G3 edges of type *Sny* and *Snn*. When this is done, we get the network G_{LS}. The construction of G_{LS} is described below.

Step1: [Construct H, the union of G_L and G_S]

Begin with a copy of G_L and one of G_S. Prefix each vertex in G_L with an l and each one in G_S with an s. This is just to make the two vertex sets different. Following this, we have the network H described above.

Step2: [Account for G1 edges as in Lemma 4]

For each $G1$ edge $<i,j>$ in N, let $<a,b>$ and $<c,d>$, respectively, be its image in G_L and

G_S.

i) Delete $<a,b>$ and $<c,d>$ from the graph.

ii) Add edges $<a,y>$, $<c,y>$, $<y,z>$, $<z,b>$, and $<z,d>$ (Figure 6). We now have the network H' described above.

Step3: [Lemmas 8 through 11 and 12 a)]

For each $G2$ edge $<i,j>$ of type *Syn* or *Snn* connect (by means of directed edges) the image of vertex j in G_S to the images in G_L of all vertices k in N such that

i) $<k,l>$ is a $G3$ edge of type *Lny* or *Lnn*;

ii) there is a path from j to k in N;

iii) $<i,j>$ is on at least one path in $longest(source(k))$;

iv) $<k,l>$ is on at least one path in $shortest(sink(j))$; and

v) all paths from j to k have the same length.

Step4: [Lemmas 8 through 11 and 12 b)]

For each $G2$ edge $<i,j>$ of type *Lyn* or *Lnn* connect (by means of directed edges) the image of vertex j in G_L to the image in G_S of all vertices k in N such that:

i) $<k,l>$ is a $G3$ edge of type *Sny* or *Snn*;

ii) there is a path from j to k in N;

iii) $<i,j>$ is on at least one path in $shortest(source(k))$;

iv) $<k,l>$ is on at least one path in $longest(sink(j))$; and

v) all paths from j to k have the same length.

Figure 7 shows the G_{LS} obtained for the network of Figure 1 using the above construction.

Lemma 13: Let P be a path in G_{LS}. P is of one of the following types:

a) All edges in P are in G_L

b) All edges in P are in G_S

c) P is of the form $P_L P_M P_R$ where all edges in P_L are in G_L; those in P_M are edges introduced in step2; and those in P_R are in G_L. Note that P_L or P_R or both may be empty.

d) P is as in c) except that all P_R edges are in G_S.

e) P is as in c) except that all P_L edges are in G_S.

f) P is as in c) except that all P_L and P_R edges are in G_S.

g) All edges in P_L are in G_S; P_M is an edge introduced in Step 3; P_R contains only edges in G_L.

h) All edges in P_L are in G_L; P_M is an edge introduced in Step 4; all edges in P_R are in G_S.

Proof: Follows from the construction of G_L, G_S, and G_{LS} and the properties of $G1$, $G2$, $G3$, $G4$ edges. □

Lemma 14: Let P be a path in G_{LS}. Let Q be its extension to a source to sink path of N. This extension is obtained in the following way:

(1) If the first (last) edge in P is in G_L, then extend leftwards (rightwards) to a source (sink) of N using a longest such extension.

(2) If the first (last) edge in P is in G_S, then the extension to a source (sink) is by a shortest such

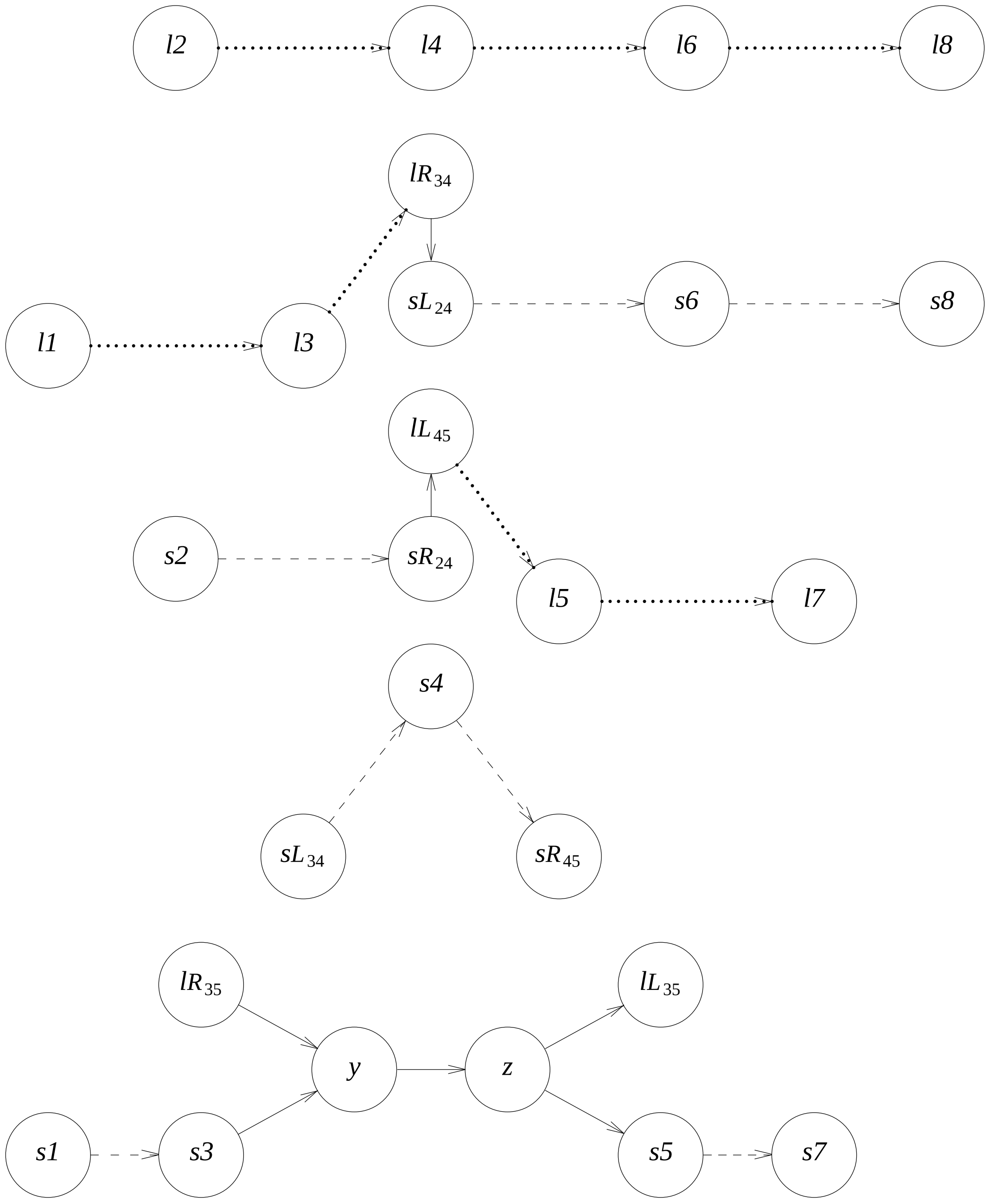
extension.

(3) If Q contains Step 2 edges, these are mapped back to the $G1$ edges of N that they are the image of.

(4) If Q contains a Step 3 or Step 4 edge, it is replaced by a j to k path in N.

Let $<i,j>$ be an edge on Q. If $<i,j>$ is in G_L, then Q *long covers* $<i,j>$. If $<i,j>$ in G_S, Q *short covers* $<i,j>$. If $<i,j>$ is a $G1$ edge, then Q both *long* and *short covers* $<i,j>$.

Proof: We consider the eight cases of Lemma 13. For P of type a), b), c) and f) the Lemma follows from the construction of G_L and G_S. For d), P_L consists of zero or more G_L edges followed by one or more Step 2 edges (actually at least 3 will be there), followed by zero or more G_S edges. Each set of 3 Step 2 edges represents a $G1$ edge. The Lemma follows from the definition of a $G1$ edge which requires that $shortest(source(i)) = longest(source(i))$ and $shortest(sink(j)) = longest(sink(j))$. The proof for e) is similar. Now consider g). P is comprised of one or more G_S edges followed by a Step 3 edge followed by one or more G_L edges. Let $<i,j>$ be the last G_S edge and let $<k,l>$ be the first G_L edge. By construction, $<i,j>$ is a $G2$ edge. Since $<i,j>$ is on at least one path in $longest(source(k))$; $shortest(source(i)) = longest(source(i))$ and all paths between j and k have the same length, it follows that the left segment of Q up to vertex k is in a path in $longest(source(k))$. Also, since $<k,l>$ is on at least one path in $shortest(sink(j))$; $shortest(sink(l)) = longest(sink(l))$; and all paths between j and k have the same length, it follows that the segment of Q from j to the sink is in $shortest(sink(j))$. The conditions of Lemma 10 are satisfied and so $<i,j>$ is short covered and $<k,l>$ long covered. From the construction of G_S and the just proved



Dotted lines are edges in G_L.
Dashed lines are edges in G_S.
solid lines are edges created by the construction.

Figure 7: G_{LS} obtained from G_L and G_S of Figures 4 and 5.

fact that the segment of Q from j to the sink is in $shortest(sink(j))$, it follows that all G_S edges in Q are short covered. Similarly, all G_L edges are long covered. The lemma is proved similarly for the case when P is of type h). □

Definition: Y is a *partial cover* of G_{LS} iff every edge of G_{LS} except possibly edges of type $e1$ through $e4$ (cf, Figure 6) introduced in Step 2 of the construction and edges introduced in Steps 3 and 4 of the construction are on at least one path in Y. □

Any set of paths that includes all dotted and dashed edges as well as the $e5$ type edge $<y,z>$ of Figure 7 defines a partial cover of the G_{LS} of Figure 7.

Lemma 15: Let Y be a *partial cover* of G_{LS}. Y is readily transformed into a MinMaxSP Z of the network N such that $|Y| = |Z|$.

Proof: The paths in Y are extended as described in Lemma 14. Each path P ε Y results in exactly one path Q ε Z. So, $|Y| = |Z|$. Further, since Y is a partial cover of G_{LS}, all edges except possibly edges of type $e1$ through $e4$ introduced into G_{LS} in Step 2 of the construction and the edges introduced in Steps 3 and 4 are included on paths in Y. From Lemma 14, it follows that the set of extended paths of N obtained from Y in the manner described in Lemma 14 long and short cover all edges of N. So, Z is a MinMaxSP of N. □

Theorem 1: Let X be a minimum cardinality MinMaxSP of N and Y a minimum cover of G_{LS}. $|X| = |Y|$.

Proof: Each path in X corresponds to exactly one path in G_{LS}. This path is obtained by simply using the mappings from N to G_L and G_S and the transformations of Steps 1 through 4 that obtain

G_{LS}. Further, the set of paths obtained in this way form a *partial cover* of G_{LS}. The size of this *partial cover* is $|X|$ nad this must be $\geq |Y|$ as Y is a minimum cover of G_{LS}. From Lemma 15 and the minimality of X, it follows that the size of the *partial cover* must exactly equal $|Y|$. □

3.4 SUMMARY

Our algorithm to obtain a MinMaxSP of a circuit, C, with rising and falling delays consists of the following steps:

S1: From C construct an equivalent network N as in [7].

S2 From N construct a *dag* G_{LS} as described in Section 3.3.

S3 Transform the *dag* G_{LS} into a network flow problem, F, as in [7]. However, Step 2 edges $e1$–$e4$, Step 3 and Step 4 edges of the G_{LS} construction have a lower capacity L_{ij} of 0 rather than 1.

S4 Find a minimum flow in F.

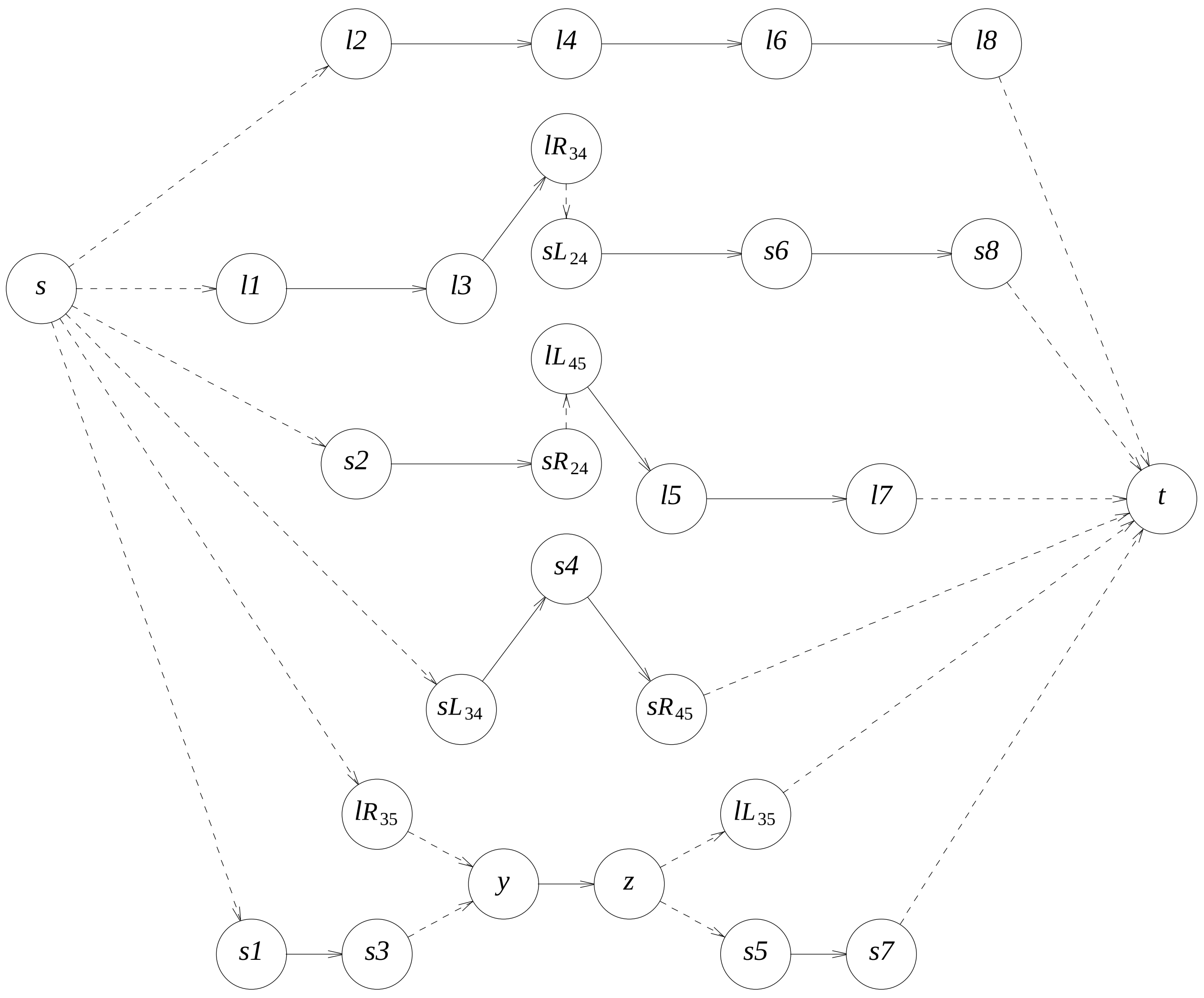
S5 From the minimum flow construct the *partial cover* of G_{LS}.

S6 From the *partial cover* obtain the MinMaxSP of N.

S7 From this obtain the MinMaxSP of C.

As in the case of [7] the overall complexity of the algorithm is dominated by S4. This step requires $O(m(m+n))$ time where n and m are, respectively, the number of vertices and edges in the circuit C.

The flow network corresponding to the G_{LS} of Figure 7 is shown in Figure 8. A *partial*



All solid lines have lower capacity 1.
All dashed lines have lower capacity 0.

Figure 8: Flow network for G_{LS} in Figure 7.

cover as well as its extension to a MinMaxSP set for the network of Figure 1 are given in Figure 9.

partial cover	extension	long covers	short covers
(s,l2,l4,l6,l8,t)	*(2,4,6,8)*	*<2,4>,<4,6>,<6,8>*	*none*
(s,l1,l3,lR_{34},sL_{24},s6,s8,t)	*(1,3,4,6,8)*	*<1,3>,<3,4>*	*<4,6>,<6,8>*
(s,s2,sR_{24},lL_{45},l5,l7,t)	*(2,4,5,7)*	*<4,5>,<5,7>*	*<2,4>*
(s,sL_{34},s4,sR_{45},t)	*(1,3,4,5,7)*	*none*	*<3,4>,<4,5>*
(s,s1,s3,y,z,s5,s7,t)	*(1,3,5,7)*	*<3,5>*	*<1,3>,<3,5>,<5,7>*

Figure 9: A partial cover of the flow network of Figure 8 and its extension with *long* and *short* covered edges.

4 EXPERIMENTAL RESULTS

We programmed our algorithm in C and experimented with the ten ISCAS circuits used in the experiments reported in [7]. Figure 10 gives the number of paths in the union of a minmum cardinality MinSP and a minimum cardinality MaxSP as well as in a minimum cardinality MinMaxSP for each of the ten circuits. The last column gives the difference between the sizes of these two sets. Figure 11 gives the run time, in seconds, on an Apollo DN3000 workstation. The time to compute the union of a minimum cardinality MinSP and a minimum cardinality MaxSP was obtained by running the algorithm of [7] to find a minimum cardinality MaxSP and then running its modification (Section 3.1) to find a minimum cardinality MinSP. The sum of these two times is the time to compute MinSP $\cup$ MaxSP. The run time of the algorithm obtained in Section 3.4 to find a minimum cardinality MinMaxSP is given in the last column.

5 REFERENCES

1 V.D. Agrawal, "Synchronous Path Analysis in MOS Circuit Simulator," *Proc. ACM IEEE*

#	circuit	$\|MinSP \cup MaxSP\|$	$\|MinMaxSP\|$	diff
1	c432	839	807	32
2	c499	1648	1520	128
3	c880	1466	1232	234
4	c1350	1768	1640	128
5	c1908	2558	2414	144
6	c2670	3739	3594	145
7	c3540	5117	5071	46
8	c5315	8702	7463	1239
9	c6288	8528	8490	38
10	c7552	10841	10651	190

Figure 10: Number of paths generated for two algorithms

#	circuit	$t(MinSP \cup MaxSP)$	$t(MinMaxSP)$
1	c432	23.23	36.50
2	c499	115.53	193.53
3	c880	79.48	152.60
4	c1350	111.17	212.87
5	c1908	268.40	391.75
6	c2670	530.60	887.52
7	c3540	992.62	1330.60
8	c5315	2587.55	4332.58
9	c6288	2032.48	2860.60
10	c7552	4672.57	6012.12

Figure 11: The run time for the two algorithms (in seconds)

19th Design Automation Conf., June 1982, pp. 629-635.

2 H.K. Al-Hussein, "Path-Delay Computation Algorithms for VLSI Systems," *VLSI Design*,

February 1985, pp. 86-91.

3 F. Brglez and H. Fujiwara, "Neutral Netlist of Ten Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. IEEE Int. Symp. Circuits & Systems*, June 1985

4 R.B. Hitchcock, Sr., "Timing Verification and the Timing Analysis Program," *Proc. ACM IEEE 19th Design Automation Conf.*, June 1982, pp. 594-604.

5 R.B. Hitchcock, G.L. Smith and D.D. Cheng, "Timing Analysis of Computer Hardware," *IBM J. Research & Development*, vol. 26, No. 1, Jan. 1982, pp. 100-108.

6 E. Lawler, *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston. 1976

7 W.N. Li, S.M. Reddy, and S. Sahni, "On Path Selection In Combinational Logic Circuits," *IEEE Transaction on CAD*, 8, 1, 1989, pp 56-63.

8 C.J. Lin and S.M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. CAD,* Sept. 1987, pp. 694-703

9 H.T. Liu and C.R. Kime, "A Delay Test Generation System for Combinational logic," *Tech Report, Dept. of Elec. & Comp. Eng, University of Wisconsin-Madison*, August 1987.

10 Y.K. Malaiya and R. Narayanaswamy, "Testing for Timing Faults in Synchronous Sequential Integrated Circuits," *Proc. 1983 Int'l. Test Conf.*, Oct. 1983, pp. 560-571.

11 E.J. McCluskey, *Logic Design Principles,* Prentine-Hall, 1986.

12 S. Patil, "An Automatic Test Pattern Generator for Delay Faults in Logic Circuits," *M.S.*

Thesis, Department of Electrical and Computer Engineering, University of Iowa, May 1987.

13 S.M. Reddy, C.J. Liu, and S. Patil, "An Automatic Test Pattern Generator for the Detection of Path Delay Faults," *Proc. Int. Conf. on Computer Aided Design*, November 1987, .pp 284-287

14 J. Savir and W.H. Mcanney, "Random Pattern Testability of Delay Faults," *Proc. 1986 Int'l. Test Cnf.*, Sept. 1986, pp. 263-273.

15 G.L Smith, "Model for Delay Faults Based Upon Paths," *Proc. 1985 Int'l. Test Cnf.*, Nov. 1985, pp. 342-349.

16 K.D. Wagner, "Delay Testing of Digital Circuits Using Pseudorandom Input Sequences," Center for Reliable Computing Report 85-12, revised March 1986, Stanford University.

[17] A.D. Friedman and P.R. Menon, Theory and Design of Switching Circuits, Computer Science Press, 1975.

[18] S.Kundu and S.M.Reddy, "On the Design of Robust Testable CMOS Combinational Logic Circuits," Proc. Int. Sym. on Fault-tolerant Computing, June 1988, pp. 220-225.

[19] S.R. Kunkel and J.E. Smith, "Optimal Pipelining in Supercomputers," 13th Annual International Symposium on Computer Architecture Proceedings, June 1986, pp. 404-412.

[20] C.J.Lin and S.M.Reddy, "On Delay fault Testing in Logic Circuits," IEEE Transactions on CAD, September 1987, pp. 694-703.

[21] K.Roy, J.A. Abraham, K. De, and S. Lusky, "Synthesis of Delay Fault Testable

Combinational Circuits," Proc. of ICCAD, November 1989, pp. 418-421.