Edge-Map: Optimal Performance Driven Technology Mapping for Iterative LUT Based FPGA Designs*

Honghua Yang and D. F. Wong

Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712

Abstract

We consider the problem of performance driven lookup-table (LUT) based technology mapping for FPGAs using a general delay model. In the general delay model, each interconnection edge has a weight representing the delay of the interconnection. This model is particularly useful when combined with an iterative re-technology mapping process where the actual delays of the placed and routed circuit are fed-back to the technology mapping phase to improve the mapping based on the more realistic delay estimation. Well known technology mappers such as FlowMap and Chortle-d only minimize the number of levels in the technology mapped circuit and hence are not suitable for such an iterative re-technology mapping process. Recently, Mathur and Liu in [ML94] studied the performance driven technology mapping problem using the general delay model and presented an effective heuristic algorithm for the problem. In this paper, we present an efficient technology mapping algorithm that achieves provably optimal delay in the technology mapped circuit using the general delay model. Our algorithm is a non-trivial generalization of FlowMap. A key problem in our algorithm is to compute a K-feasible network cut such that the circuit delay on every cut edge is upper-bounded by a specific value. We implemented our algorithm in a LUT based FPGA technology mapping package called Edge-Map, and tested Edge-Map on a set of benchmark circuits.

1 Introduction

Technology mapping of a Boolean circuit for lookup-table (LUT) based field programmable gate array (FPGA) designs is to convert the circuit into a functionally equivalent network comprised of LUTs. The technology mapping phase of FPGA designs is preceded by a technology independent logic optimization phase, and followed by placement and routing phases. Since the routing resources in FPGAs are more limited and slower compared with traditional ASIC technologies, technology mapping has profound impact on the placement and routing phases and hence chip performance. Minimizing circuit delay has been a goal for many previous technology mapping algorithms. Previous performance driven technology mapping algorithms for LUT-based FPGAs have focused on minimizing the number of levels in the solution and minimizing the estimated circuit delay in the solution. The algorithms for minimizing the number of levels include Chortle-d [FRV91] which minimizes circuit level increase at each bin packing step, and FlowMap [CD92] which gives an optimal mapping solution with the minimum number of levels. However, the number of levels in a technology mapped circuit does not accurately reflect the delay in a placed and routed FPGA chip, since the net delays between LUTs are in general non-uniform in the final design. The heuristic algorithms for minimizing the estimated circuit delay include MIS-pga-delay [MSBSV91] which combines technology mapping with layout synthesis and resynthesis to minimize circuit delay, and more recently heuristic mapping algorithms for delay minimization with feed-backs from placement and routing [CTC+93] and Bias-Clus in [ML94]. FlowMap [CD92] can be modified to handle the delay model that allows different delays for different nets, but it requires the same delay for all interconnections in the same net.

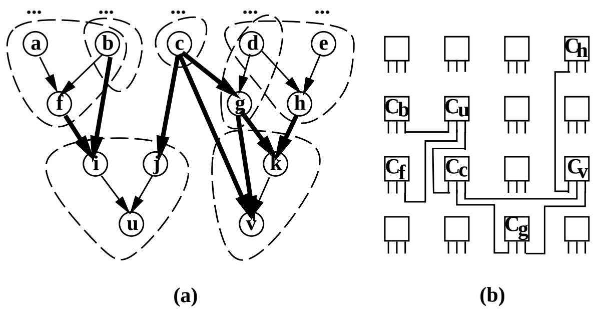

Figure 1: (a) A part of a technology mapped circuit. Each circled area represents a feasible cone. (b) The placed and routed circuit. C_v denotes the LUT implementing the cone rooted at v.

A general delay model was proposed by Mathur and Liu [ML94] that allows different delays for different interconnections even if they belong to the same net. This delay model reflects the fact that the delays of the interconnections in the same net may differ significantly because of the positions of the logic blocks, the limited routing resources in an FPGA chip, and the substantial differences in delays among the available routing resources. For example, in Figure 1 different

*This work was partially supported by the Texas Advanced Research Program under Grant No. 003658459, by an Intel Foundation Graduate Fellowship, by a DAC Design Automation Scholarship, and by a grant from AT&T Bell Laboratories.

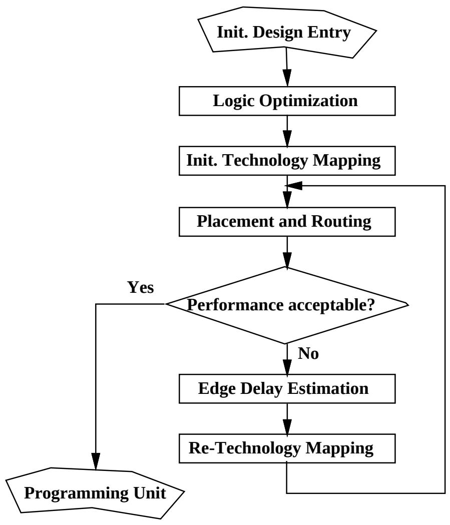

Figure 2: A typical CAD system for FPGAs with iterative re-technology mapping.

branches of the same net at the output of LUT C_c are routed along different paths with significantly different delays. This model is particularly useful when combined with an iterative re-technology mapping process where the actual delays of the placed and routed chip are fed-back to the technology mapping phase to guide the re-technology mapping based on the more realistic delay estimation. Figure 2 shows the steps involved in a typical CAD system with iterative re-technology mapping for implementing a circuit in an FPGA. This kind of CAD systems requires a technology mapping algorithm which can consider interconnection delay information. Existing performance driven technology mappers such as FlowMap assume unit delay for all branches in a net and hence cannot be used in the above iterative approach.

A heuristic technology mapping algorithm Bias-Clus for minimizing circuit delay on this general delay model was presented in [ML94]. Bias-Clus uses a *biased depth first search* to form "long" clusters along the most critical paths, thus reducing delay on those paths. Bias-Clus was implemented with an existing placement and routing system, and the technology mapping solutions produced by Bias-Clus were compared with those produced by other algorithms. In terms of the delay of the placed and routed circuit, Bias-Clus outperforms FlowMap, showing that minimizing the number of levels in the technology mapped circuit does not ensure a low delay realization, and that it is crucial to take into consideration some estimate of the actual delay incurred when a particular edge is exposed in the technology mapping process.

In this paper, we present an efficient technology mapping algorithm that achieves provably optimal delay in the technology mapped circuit using the general delay model. Our algorithm is an improvement over the previous heuristic approach in Bias-Clus for the problem since Bias-Clus only reduces the delay on the most critical paths and thus some previously non-critical paths might become critical to offset the gain obtained on the most critical paths. Our algorithm is different from the previous technology mapping algorithms for delay minimization in that we label circuit delays on edges as well as on nodes. For example, FlowMap assigns labels to nodes that represent level numbers in the LUT network. Using the edge delay labeling, we are able to solve the technology mapping problem optimally on the more general delay model by developing a new network flow technique that finds a special network cut that avoids cutting through edges with large circuit delays. Such special network cut has the property that the circuit delays on all the cut edges are upper-bounded by a specified value.

The rest of the paper is organized as follows. In Section 2, we formally define our problem and the general delay model that allows different delays on different edges. Section 3 presents the optimal technology mapping algorithm Edge-Map using the general delay model. We show some experimental results in Section 4, and conclude the paper in Section 5.

2 Problem Formulation

A *combinational Boolean circuit* is represented by a directed acyclic graph (DAG) $N = (V, E)$, where V is the set of nodes representing gates and (PI) nodes, and E is the set of directed edges representing interconnections between gates. A circuit is K-*bounded* if the number of inputs to any node is no more than K. The circuit in Figure 1 (a) is 3-bounded. In the rest of the paper, we consider K-bounded circuits only[1]. A K-*feasible cone* rooted at v, denoted by C_v is a subgraph of N consisting of v and some of its predecessors such that any path connecting a node in C_v to v lies entirely in C_v, and that the number of *distinct* input nodes to C_v is no more than K.

We assume that each configurable logic block in an FPGA is a K-input 1-output lookup-table (K-LUT), where K usually ranges from 3 to 9. The *technology mapping problem* for K-LUT based FPGA design is to cover the circuit with K-feasible cones such that every node in the circuit is contained in one (or more) of the K-feasible cones, and that each input to a cone is an output of another cone. In Figure 1 (a), each circled area represents a feasible cone. A *technology mapping solution* S_N of the circuit N is a DAG where each node represents a K-feasible cone (or equivalently K-LUT) in N, and an edge (C_u, C_v) exists if u is an input node to C_v in N. An edge of the original circuit N is *visible* if it is also in the edge set of the technology mapping solution S_N. The thick edges in Figure 1 (a) are the visible edges. Note that the covering cones in a technology mapping solution can be overlapping in order to minimize delay, which means the nodes in the overlapped area are duplicated.

The delay of an FPGA circuit depends on two factors: the delay in the K-LUTs and the delay in the interconnections (i.e., visible edges). Since the access time of a K-LUT is a constant independent of the function it implements, we can ignore the constant delay of a K-LUT by adding the constant to the delays of the visible edges and then consider edge delays only.

[1]If a given circuit is not K-bounded, there are a few algorithms such as tech-decomp in [BRSV87] and DMIG in [CCD$^+$92] to transform it into a K-bounded circuit.

Note that this modification will not change the delay along any path in a technology mapping solution.

We assume that each edge e in N has an estimated delay given by $\delta(e)$. Given a technology mapping solution S_N of the circuit N, the *circuit delay at a node* v in N with respect to the mapping solution S_N, denoted by $d(v)$, is the maximum total delay of the visible edges along any path from a PI node to v; the *circuit delay at an edge* $e = (u, w)$ in N with respect to S_N, denoted by $d(e)$, is the maximum total delay of the visible edges along any path from a PI node to the ending node w of e. Hence for every node v and every edge $e = (u, w)$ in N, if C_v is a K-feasible cone rooted at v in the mapping solution S_N, then $d(e)$ and $d(v)$ can be computed by the following two recurrence equations. Initially, $d(i) = 0$ for every PI node i.

$$d(e) = d(u) + \delta(e), \text{ where } u \text{ is the starting node of } e. \quad (1)$$
$$d(v) = \max\{d(e) \mid \text{where } e \text{ is an input wire to } C_v\}. \quad (2)$$

A *critical path* in N with respect to the technology mapping solution S_N is a path starting at a PI node and ending at a PO node o such that $d(o)$ is maximum among all PO nodes. A technology mapping solution is *optimal*, if the delay of a critical path is minimum among all technology mapping solutions.

3 An Optimal Performance Driven Technology Mapping Algorithm

From Equations (1) & (2), we can see that an optimal k-feasible cone of a node can be constructed using some of the optimal cones of its predecessors. This overall strategy is similar to that used in FlowMap. FlowMap assigns labels to nodes that represent level numbers in the LUT network, collapses nodes with the maximum labels, and then finds a network cut to be a level optimal cone for a node. However, our algorithm is a non-trivial generalization of FlowMap. Our delay optimal technology mapping algorithm is different from FlowMap in that in order to deal with the different edge delays on the edges outgoing from the same node, we label delays on edges as well as on nodes, and then repeatedly find a network cut for which *the circuit delay on every cut edge is bounded.* We design a novel node-splitting technique to handle the delay bound on the cut edges.

Our algorithm consists of two phases: *the labeling phase* and *the generating phase.* In the labeling phase, we first compute a K-feasible cone for each node v that would give v an optimal delay in the technology mapping solution. We process the nodes according to their topological ordering[2], from the PI nodes to the PO nodes. The generating phase is very simple. We generate the K-LUTs computed in the labeling phase in a bottom-up fashion, by first generating the K-LUTs for the PO nodes, and then generating the K-LUTs for the input nodes of the already generated K-LUTs.

3.1 The Labeling Phase

When computing a K-feasible cone for a node v, it suffices to consider the sub-circuit N_v consisting of v and all its predecessors (see Figure 3 (a)), since the delay at v, $d(v)$, depends only on the delays and the K-feasible cones of its predecessors, and for all predecessors u of v (except v) the delays $d(u)$ and the K-feasible cones C_u have already been computed since they all precede v in the topological ordering. For each edge e in N_v, $d(e)$ can be easily computed by $d(e) = d(u) + \delta(e)$, where u is the starting node of e. Note that $d(e)$ are not necessarily increasing along the edges in a directed path in N_v (see Figure 4 (a)), if $\delta(e)$ varies significantly on different edges[3]. If a placement algorithm chooses to place the PO nodes first and works toward the PI nodes, then it is often the case that $\delta(e)$ on the edges closer to the PI nodes are much larger than those on the edges closer to the PO nodes.

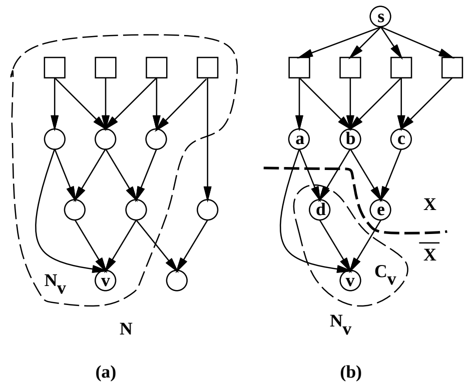

Figure 3: (a) A 3-bounded circuit N and a sub-circuit N_v. (b) N_v augmented with s. A cut $(X, \bar{X})$. $\bar{X}$ corresponds to a 3-feasible cone C_v.

We use network maximum flow techniques to compute an optimal K-feasible cone for v, and develop a novel node-splitting technique to handle different edge delays and bounded edge delays when minimizing delay for v. We augment the single-sink DAG N_v with a super source s. We first introduce some terminologies on a single-source s and single-sink v digraph N_v. A *cut* $(X, \bar{X})$ of N_v is a partition of nodes in N such that $s \in X$ and $v \in \bar{X}$. An edge whose starting node (ending node) is in X and ending node (starting node) is in $\bar{X}$ is called a *forward edge (backward edge).* The edge cut $e(X, \bar{X})$ induced by the cut $(X, \bar{X})$ is the set of forward edges of N_v. The node cut $n(X, \bar{X})$ induced by the cut $(X, \bar{X})$ is the set of nodes of X that are the starting nodes of the forward edges in $e(X, \bar{X})$. In Figure 3 (b), $e(X, \bar{X}) = \{(a, v), (a, d), (b, d), (e, v)\}$, and $n(X, \bar{X}) = \{a, b, e\}$. The *node cut size* of $(X, \bar{X})$ is the number of nodes in $n(X, \bar{X})$, denoted by $|n(X, \bar{X})|$. The *edge cut size* of $(X, \bar{X})$ is the number of forward edges in $e(X, \bar{X})$, denoted by $|e(X, \bar{X})|$. A cut is a *min-node cut* if it has the minimum node cut size among all cuts. A cut is a *min-edge cut* if it has the minimum edge cut size among all cuts.

[2]A topological ordering of all nodes in a circuit N is a linear ordering L of the nodes such that node i appears before node j in L if and only if i is a predecessor of j in N.

[3]However, the $d(e)$ values on the visible edges along any path in a mapping solution is non-decreasing because of Equations (1) & (2).

Note that given a cut $(X, \bar{X})$ of node cut size $\leq K$ in N_v, $\bar{X}$ corresponds to a K-feasible cone C_v, $n(X, \bar{X})$ corresponds to the set of input nodes to C_v (see Figure 3 (b)), and the circuit delay at v is given by

$$d(v) = \max\{d(e) | e \text{ is an edge in } e(X, \bar{X})\}.$$

This is because $d(e) = d(u) + \delta(e)$ where u is the starting node of the edge e, represents the accumulated visible edge delays at e. In order to minimize $d(v)$, we need to find a cut $(X, \bar{X})$ such that $|n(X, \bar{X})| \leq K$ and that the maximum delay $d(e)$ at the cut edges e in $e(X, \bar{X})$ is minimized. We sort the values of $d(e)$ for all edges e in N_v (whether visible or not) into non-decreasing order:

$$d_1 \leq d_2 \leq \ldots \leq d_k \leq d_{k+1} \leq \ldots.$$

We want to find the smallest d_k such that there is a cut $(X, \bar{X})$ in N_v with node cut size $|n(X, \bar{X})| \leq K$, and that for any cut edge $e \in e(X, \bar{X})$, $d(e) \leq d_k$. Once we find such d_k and $(X, \bar{X})$, then $C_v = \bar{X}$ and $d(v) = d_k$ is the optimal delay for v.

Let us denote a cut $(X, \bar{X})$ such that $\forall e \in e(X, \bar{X})$, $d(e) \leq d$, by *d-cut* (see Figure 4 (a)). A d-cut is a cut such that the circuit delays on the cut edges are bounded by d. We call a d-cut $(X, \bar{X})$ with node cut size $\leq K$ a *K-feasible d-cut.* We first need to solve a sub-problem:

Given d, find a K-feasible d-cut in N_v.

A d-cut is a *min-node d-cut* if it has the minimum node cut size among all d-cuts in N_v. The sub-problem can be reformulated as follows:

Given d, is the node cut size of
a min-node d-cut in $N_v \leq K$?

3.2 Finding A Min-Node d-Cut

An *augmenting path* from u to v in a flow network is a simple path from u to v in the undirected graph resulted from the network by ignoring edge directions that can be used to push additional flow from u to v. The *capacity of a cut* $(X, \bar{X})$ is the sum of the capacities on the *forward edges* in $e(X, \bar{X})$.

The max-flow min-cut theorem [FF62] says that, given a max-flow f in a flow network G, let $X = \{v :$ there is still an augmenting path from s to v in $G\}$, and let $\bar{X}$ be the set of the rest of the nodes in G. Then $(X, \bar{X})$ is a cut of minimum capacity (which is equal to $|f|$), and f saturates all forward edges in $e(X, \bar{X})$. If all cut edges have unit capacity, then the edge cut $e(X, \bar{X})$ is also a min-edge cut. We develop a novel node-splitting technique to reduce the minimum node cut size constraint to the minimum edge cut size constraint, and to handle the bounded delay constraint d on the cut edges.

We construct a flow network N'_v from N_v by node-splitting as follows (see Figure 4 (a) & (b)):

1. For each node u in N_v except s and v, define two nodes u_1 and u_2 with a *bridging edge* (u_1, u_2) in N'_v.
2. For each edge $e = (u, w)$ with $d(e) \leq d$, define an edge (u_2, w_1) in N'_v.
3. For each edge $e = (u, w)$ with $d(e) > d$, define an edge (u_1, w_1) in N'_v.

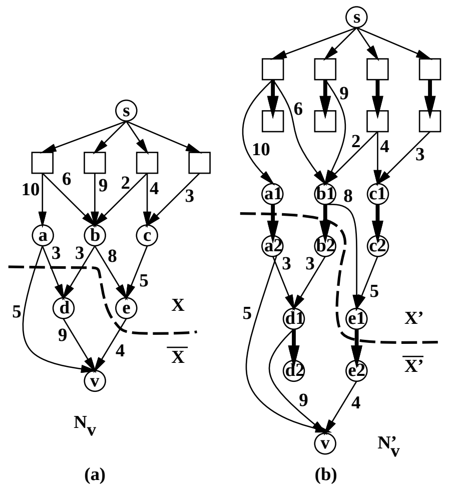

Figure 4: (a) The labels on the edges are $d(e)$ values, and $(X, \bar{X})$ is a 5-cut. (b) N'_v is obtained after node-splitting with $d = 5$. The thick edges (i.e., bridging edges) have unit capacity. The other edges have infinite capacity.

4. For the exceptional cases involving the source s and the sink v in 2 & 3, if $u = s$ then $u_2 = s$, if $w = v$ then $w_1 = v$.
5. Assign *unit capacity* to all bridging edges and *infinite capacity* to all other edges in N'_v.

Given a cut $(X, \bar{X})$ in N_v, we define its *corresponding cut in N'_v* to be a cut $(X', \bar{X}')$ such that $\bar{X}'$ contains all nodes split from the nodes in $\bar{X}$ (including v) as well as the nodes u_2 where u is in the node cut $n(X, \bar{X})$, and X' contains the rest of the nodes in N'_v. In Figure 4, the cut $(X', \bar{X}')$ in (b) is the corresponding cut of the cut $(X, \bar{X})$ in (a). Given a cut $(X', \bar{X}')$ in N'_v, if all forward edges in $e(X', \bar{X}')$ are bridging edges, we can similarly define its corresponding cut in N_v. By this definition, an edge (b, d) in N_v is a cut edge in $e(X, \bar{X})$ if and only if $b_1 \in X'$ and $d_1 \in \bar{X}'$. Note that a min-edge cut in N'_v contains bridging edges only, since by the max-flow min-cut theorem the minimum capacity of an edge cut equals a maximum total flow (with a finite flow value) in N'_v, and we assign unit capacity to bridging edges and infinite capacity to others. Further, the min-edge cut size equals the capacity of the min-edge cut.

Also note that existing node-splitting techniques used in [CD92] and [YW94] do not distinguish edges by their delays, and treat all edges the same as in item 2 during node-splitting. Using the existing node-splitting techniques, any edge (b, d) in N_v can become a cut edge by cutting through the bridging edge (b_1, b_2) in N'_v, since removing (b_1, b_2) would separate b_1 from d_1 (see Figure 4). Our new node-splitting technique treats edges differently in items 2 & 3 according to their delays, and by defining an edge (b_1, e_1) in N'_v for an edge (b, e) in N_v with delay $> d$, cutting through the bridging edge (b_1, b_2) will not separate b_1 and e_1 and hence (b, e) will not be a cut edge in N_v. We show in Lemma 3.1 that this technique effectively prevents an edge with delay $> d$ from becoming a cut edge in a cut we find in N_v.

Lemma 3.1 *$(X, \bar{X})$ is a d-cut in N_v iff its corresponding cut $(X', \bar{X'})$ in N'_v contains bridging edges only in its edge cut. Further, $|n(X, \bar{X})| = |e(X', \bar{X'})|$.*

Our sub-problem can again be reformulated as follows:

Is the edge cut size of a min-edge cut in $N'_v \leq K$?

This problem can be solved by a direct application of the max-flow min-cut theorem.

3.3 The Complete Algorithm

We now give the complete Edge-Map algorithm and prove its optimality.

```
Algorithm Edge-Map
Input: A circuit $N = (V, E)$, a delay function $\delta : E \to \Re$, and $K$.
Output: An (delay) optimal mapping solution $S = (V_S, E_S)$.
begin
/* The labeling phase */
0. Assign 0 to $d(u)$ for each PI node $u$;
1. Sort the nodes in $V$ by topological order into a list $V_T$;
2. for each node $v$ in $V_T$ in topological order
3.     Construct $N_v$;
4.     Compute $d(e) = d(u) + \delta(e)$ for each edge $e = (u, w)$ in $N_v$;
5.     Sort the different values of $d(e)$ for all edges $e$ in $N_v$:
       $d_1 < d_2 < \cdots < d_k < d_{k+1} < \cdots$;
6.     $k = 1$; $done = false$;
7.     while not done
8.             Find a a min-node $d_k$-cut $(X, \bar{X})$ in $N_v$;
               if $|n(X, \bar{X})| > K$ then $k = k + 1$;
               else $done = true$;
/* The generating phase */
9. $V_S = \phi$; $L$ = the list of PO nodes in $N$;
10.while $L$ is not empty
11.        Remove a node $v$ from $L$;
12.        $V_S = V_S \cup \{C_v\}$; $L = L\cup$ {all input nodes to $C_v$};
```

We first present 2 lemmas that are necessary in proving Theorem 3.1.

Lemma 3.2 *For any node v in N_v, $d(v) \leq$ the optimal delay $d_{opt}(v)$ of v in an optimal K-LUT technology mapping solution.*

Lemma 3.3 *For any node v in N, if C_v is generated in the generating phase, then the delay of v in the technology mapping solution is $\leq d(v)$.*

Theorem 3.1 *Algorithm Edge-Map produces an (delay) optimal K-LUT technology mapping solution, and it runs in $O(Knm \log m)$ time where n, m are the numbers of nodes and edges in the circuit, respectively.*

Proof: Lemma 3.2 proves the optimality of the labeling phase, and Lemma 3.3 shows that the K-feasible cones generated in the generating phase are an optimal technology mapping solution.

By the max-flow min-cut theorem [FF62], the minimum capacity of an edge cut (i.e., the size of a min-edge cut since a minimum capacity edge cut can have only bridging edges with unit capacity) is equal to a maximum total flow in N'_v. Since we are only interested in finding a min-edge cut with size no more than K in N'_v, and each augmenting path in N'_v from s to v increases the flow by one unit (since a bridging edge has unit capacity), we only need to find at most $K+1$ augmenting paths in N'_v to compute a maximum flow in step 8 (also see [CD92]). If we can find $K+1$ augmenting paths in N'_v, then the maximum flow value and the min-edge cut size in N'_v is at least $K+1$, and there is no need to compute the exact maximum flow in N'_v.

Since finding an augmenting path takes $O(m)$ time, finding a min-edge cut with edge cut size no more than K takes $O(Km)$ time. Hence steps 8 takes $O(Km)$ time. We can use a binary search strategy in the loop in step 7 to search for the smallest k such that there exists a min-node d_k-cut of node cut size $\leq K$. Hence steps 7-8 takes $O(Km \log m)$ time. Since we compute the optimal delay for each node in N in topological order at step 2, the whole algorithm takes $O(Knm \log m)$ time. $\square$

Note that FlowMap takes $O(Knm)$ time. Hence our Edge-Map algorithm is only slower than FlowMap by a factor of $\log m$ in the worst case using the general delay model. As a special case, Edge-Map can generate the same results as FlowMap, by setting $\delta(e) = 1$ for all edges in N.

3.4 Post-Processing

In the post-processing phase, we want to reduce the number of LUTs in the (delay) optimal solution without increasing the optimal circuit delay. Although making a previously non-visible edge visible would possibly increase the delay in an optimal mapping solution, packing an *entire* LUT into its fanout LUT will not increase the delay in an optimal mapping solution since the $d(e)$ on the visible edges along any path in the solution is non-decreasing because of Equations (1) & (2).

We discovered that for a given LUT with output v, we can apply Flow-Pack [CD92] operations to pack a set of the predecessor LUTs of LUT v into a single K-LUT. Flow-Pack operations will not increase the circuit delay in our optimal mapping solution since they pack multiple (entire) LUTs into their fanout LUTs.

4 Experimental Results

We have implemented our Edge-Map algorithm in a version of SIS [BRSV87] that contains the FlowMap package [CD92], using the C language on Sun SPARC workstations. We tested Edge-Map on a set of ISCAS and MCNC benchmark circuits that were decomposed into 2-input circuits of simple gates using *tech-decomp* and *dmig*.

Initially, we assigned every edge in a net an estimated delay based on the number of nodes contained in the net, and applied the Edge-Map algorithm to obtain a delay optimal mapping solution for the initial delay estimation. We then applied a performance driven placement program on the mapped circuit to obtain delay estimation for the visible edges. This delay information was re-assigned to the visible edges of the technology mapping solution.

There are several strategies to assign delay estimation to the invisible edges in a LUT in the previous mapping solution. (1) Each invisible edge in a LUT receives a delay estimation 0. This strategy was adopted by Bias-Clus. (2) Each invisible edge in a LUT receives a delay estimation equal to the *average delay* of the visible edges incoming to the LUT and the visible edges outgoing from the root of the LUT. (3) Each invisible edge in a LUT receives a delay estimation equal to the *maximum delay* of the visible edges incoming to

[illegible]	[illegible]	[illegible]			[illegible]	
		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]				[illegible]		

[illegible]