# Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-Driven Bipartitioning *

Andrew B. Kahng
CSE and ECE Departments
UCSD
La Jolla, CA 92093
abk@ucsd.edu

Xu Xu
CSE Department
UCSD
La Jolla, CA 92093
xuxu@cs.ucsd.edu

## ABSTRACT

Traditional multilevel partitioning approaches have shown good performance with respect to cutsize, but offer no guarantees with respect to system performance. Timing-driven partitioning methods based on iterated net reweighting, partitioning and timing analysis have been proposed [2], as well as methods that apply degrees of freedom such as retiming [7] [5]. In this work, we identify and validate a simple approach to timing-driven partitioning, based on the concept of "$V$-shaped nodes". We observe that the presence of $V$-shaped nodes can badly impact circuit performance, as measured by maximum hopcount across the cutline or similar path delay criteria. We extend traditional KLFM approaches to directly eliminate or minimize "distance-$k$ $V$-shaped nodes" in the bipartitioning solution, achieving a smooth trade-off between cutsize and path delay. Experiments show that in comparison to MLPart [4], our method can reduce the maximum hopcount by 39% while only slightly increasing cutsize and runtime. No previous method improves path delay in such a transparent manner.

## Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*Design Aids*; J.6 [**Computer Applications**]: COMPUTER-AIDED ENGINEERING

## General Terms

Algorithms, Performance, Design

## 1. Introduction

With increased system complexity, circuit hypergraph partitioning, which is the "divide" step of the divide-and-conquer paradigm, plays a crucial role in many design tasks [5]. The problem is to divide the nodes of a graph into several roughly equal parts; the traditional objective is to minimize cutsize. Among many partitioning methods, multilevel approaches (e.g., MLPart [3] or hMetis [12]) are considered effective for cutsize minimization. Such methods perform a sequence of netlist coarsening and uncoarsening steps with FM-based partitioning refinement at each level of the uncoarsening hierarchy. While these algorithms outperform other algorithms in cutsize, they cannot guarantee to produce small delay in general.

For performance-driven contexts, the hypergraph partitioner must consider the impact of implied interconnects[1] on performance. The primary objective of a *performance-driven partitioner* is to minimize path delay on timing paths. Recently, several performance-driven partitioning methods have been proposed. Most of these methods do not consider cutsize, and no smooth cutsize-delay tradeoff (let alone transparent consideration of path delay within traditional cutsize-driven approaches) has been discovered. In particular, existing performance-driven partitioners either try to modify the input of multilevel FM partitioners, by means such as reweighting [2], or else apply novel approaches such as min-delay clustering [7]. However, these approaches may be impractical because of large cutsize [6] or large runtime [2]. Furthermore, improvements in timing are often not obvious. The goal of our work is to find a performance-driven partitioner that can provide a smooth cutsize/delay tradeoff. Our contribution is summarized as follows.

1. We define the concept of a *$V$-shaped node* in a partitioning solution, as well as its generalization to *distance-$k$ $V$-shaped node*. We observe that even a few $V$-shaped or distance-$k$ $V$-shaped nodes in the partitioning solution may significantly increase path hopcounts across the cutline. This suggests improving performance by eliminating such nodes.

2. We propose a new algorithm to eliminate, or at least reduce, $V$-shaped nodes. Instead of modifying the input of MLPart, we modify the MLPart algorithm itself by changing the gain function. We also use a "look-ahead" algorithm reminiscent of CLIP [9] to eliminate distance-$k$ $V$-shaped nodes.

3. Our method is easily implementable within standard FM with little cutsize and runtime penalty. We focus

*This research was partially supported by the MARCO Gigascale Silicon Research Center.


*ISPD'03,* April 6–9, 2003, Monterey, California, USA.


[1]That is to say, any cut net will correspond to an interblock wire, or a wire that has some expected length that depends on the size of the given partitioning instance.

on the flat bipartitioning engine context, but our result can be applied within multilevel or any other framework that invokes standard FM; our approach also extends easily to multiway implementations. Our experimental results show that the method can achieve an average of 39% hopcount reduction on industry testcases, with negligible implementation effort and negligible impact on cutsize and runtime.

## 2. Notation and Problem Formulation

Below, we use the following notation.

- $H(V, E)$ denotes the circuit hypergraph.
- $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes representing components (e.g., cells) of the circuit.
- $E = \{e_1, e_2, \ldots, e_m\}$ is the set of signal nets, where each net is a subset of nodes that are electrically connected by a signal.
- $V_c$ is the subset of combinational nodes, and $V_s$ is the subset of sequential nodes (or *FF nodes*); $V_c \bigcup V_s = V$ and $V_c \bigcap V_s = \phi$.
- A *bipartition* $(V_0|V_1)$ of $H(V, E)$ divides $V$ into two disjoint subsets $V_0$ and $V_1$, such that $V = V_0 \bigcup V_1$; the two subsets are also called *Part 0* and *Part 1*.
- For a net $e = \{v_1, ..., v_l\}$ where $v_1$ is the fanout node whose output signal is the input signal to $v_j$ $(2 \leq j \leq l)$, we say that $v_1$ is the *input* of $v_j$ $(2 \leq j \leq l)$, and that each $v_j$ $(2 \leq j \leq l)$ is an *output* of $v_1$.
- If node $v_i$ is an input of node $v_j$, then we say that there is a *directed edge* from $v_i$ to $v_j$.
- *PI* denotes the set of primary inputs, and *PO* denotes the set of primary outputs. For purposes of path timing analysis we treat the nodes of *PI*, *PO* and *FF* as the end points of timing paths, i.e., the *circuit delay* is the longest combinational path delay from any FF or PI output to any FF or PO input. We generically refer to timing paths as *FF-FF* paths.
- $P = (v_{p_1}, ..., v_{p_l})$ is a *directed path* from $v_{p_1}$ to $v_{p_l}$ if there exists a directed edge from $v_{p_{j-1}}$ to $v_{p_j}$, $(2 \leq j \leq l)$. We say that the *length* of $P$ is $l - 1$.
- Let $P = (v_{p_1}, ..., v_{p_l})$ be a directed path from $v_{p_1}$ to $v_{p_l}$. If $v_{p_j} \in V_c$, $(2 \leq j \leq l-1)$, $P$ is a *combinational directed path*.
- Let $P = (v_{p_1}, ..., v_{p_l})$ be a directed path from $v_{p_1}$ to $v_{p_l}$. If $v_{p_1}, v_{p_l} \in V_s$ and $v_{p_j} \in V_c$, $(2 \leq j \leq l-1)$, $P$ is a *FF-FF path*.
- A combinational node $v_i \in V_c$ is a *distance-k V-shaped node*, or *$V_{(k)}$-node*, if it satisfies (1) $\exists$ $v_j$ , $v_t$ such that there is a directed edge from $v_j$ to $v_i$ and a combinational directed path from $v_i$ to $v_t$ whose length is $k$; and (2) $v_j$ and $v_t$ are not in the same part of $v_i$. For the special case of $k = 1$, $v_i$ is called a *V-shaped node*.
- $A$ = the total area of all the nodes in $V$. $A_0$ (resp., $A_1$) = the total area of all the nodes in $V_0$ (resp., $V_1$).
- If $\exists$ $v_i, v_j \in e$ such that $v_i \in V_0$ and $v_j \in V_1$, then $e$ is a *cut hyperedge* of the bipartition $(V_0|V_1)$.
- The *cutset* of a bipartition is $E((V_0|V_1)) = \{e \in E \mid e$ is a cut hyperedge of $(V_0|V_1)\}$. The *cutsize* of the bipartition is $|E((V_0|V_1))|$.
- A directed edge from $v_i$ to $v_j$ is a *hop* if $v_i$ is the input to $v_j$ in a cut hyperedge.
- $h(P)$ denotes the *hopcount* of a *FF-FF* path $P$, i.e., the number of hops in $P$.
- $h$ = the maximum value of $h(P)$ over all FF-FF paths in $H(V, E)$.
- A *critical path* is a FF-FF path whose hopcount is equal to $h$.
- Suppose a bipartition $(V_0|V_1)$ has a *cut* on $e$ and let $v_i$ be the fanout node whose output signal is the input to the rest of the nodes in the net. If $v_i \in V_0$, the *cut direction* is indicated as $i$; if $v_i \in V_1$, the cut direction is indicated as $o$.

**Performance Driven Bipartition Problem (PDBP)**

**Given:**
Hypergraph $H = (V, E)$
Area balance tolerance $s$ $(0 < s < 1)$, a given parameter that constrains partition areas
$\alpha$, a given parameter which captures the desired tradeoff between cutsize and path delay in the objective function
**Find :**
A bipartition $(V_0|V_1)$ which satisfies

$$(1 - s) \cdot \frac{A}{2} \leq A_0 \leq (1 + s) \cdot \frac{A}{2}$$

and minimizes

$$\alpha|E((V_0|V_1))| + (1 - \alpha)h((V_0|V_1))$$

## 3. Previous Performance Driven Methods

Most previous performance-driven partitioning approaches alter the netlist using logic replication, retiming or buffer insertion to meet delay constraints while minimizing the cutsize [7] [5] [6] [13] [8]. For example, Cong et al. [7] propose a global clustering-based partitioning algorithm. The basic idea is:

- Construct a clustered circuit with the minimum clock period, and perform retiming and node duplication as possible.
- Perform cutsize driven clustering on the clusters formed in the previous step.
- Perform simultaneous cutsize and delay refinement during cluster decomposition.

The method reduces delay by 16% while increasing cutsize by 17% – *with retiming* – compared with hMetis [7]. However, such methods can require substantial gate replication, which potentially increases die area. Some of these methods

[6] tend to produce noticeably worse cutsize compared to multilevel FM partitioning.

Other approaches [11] [2] have been proposed which do not change the netlist. Typically, a multilevel FM partitioner such as hMetis [12] is used with some modified (weighted) input. In the taxonomy of [2], these approaches can be divided into *net-based* and *path-based* categories. Net-based partitioning approaches ([**?**], according to [2]) define a criticality value for each net after timing analysis, while path-based approaches consider the criticality of paths instead of single nets [2]. All of these methods require timing analysis in order to find critical nets or paths, and then reweight the critical nets or paths so as to reduce the chances of cuts occurring. According to Ababei et al. [2], their bipartitioning algorithm can reduce delay by 14% at the expense of an increase of 10% in cutsize and 139% in runtime, compared with hMetis [2].

We observe that timing analysis can take a long time and that it is necessarily based on an inaccurate delay model. Delay models such as that of Cong et al. [7] (node delay = 1, intra-block delay = 0, and inter-block delay = 5) may identify "critical paths" that incorrectly drive the timing analysis and, hence, the partitioner. Since we may not have enough information at the partitioning stage to make accurate delay estimates, for some testcases these algorithms can produce partitioning solutions with worse delay than solutions found by generic multilevel FM partitioning.[2]

Recently, some algorithms have been proposed which attempt to incorporate timing analysis results somewhat more directly into the FM partitioner [1]. For example, Ababei et al. [1] propose to perform timing analysis first in order to assign a "criticality" value to each edge. Then, the gain function of the FM partitioner is changed such that edges with higher criticality will not be cut. Since criticality is a global variable, this means that to obtain a reasonable value of criticality, global timing analysis is needed. Moreover, since the change of one node in the partitioning solution may affect the criticality values of many other nodes, the complexity and convergence of the approach become difficult, as witnessed by the level of improvements reported.

# 4. Solving PDBP via Elimination of $V_{(k)}$-nodes

In a bipartition $(V_0|V_1)$, if all nets in the cutset have the same cut direction, then we call $(V_0|V_1)$ a *unidirectional bipartition*. A key intuition stems from the fact that in a unidirectional bipartition, the hopcount of any FF-FF path is at most one. So, unidirectional bipartitioning is in some sense an "idealized goal" for performance-driven partitioning, and can be sought by, e.g., flow-based methods. Unfortunately, a unidirectional bipartition tends to have much higher cutsize than multilevel FM solutions, and for some testcases no purely unidirectional solution exists. Therefore, we propose to relax the unidirectional condition to "locally unidirectional". We call a bipartition $(V_0|V_1)$ without any $V_{(k)}$-nodes (defined in Section 2 above) as a *distance-k unidirectional bipartition*. Our intuition is that a smooth tradeoff between cutsize and delay can be achieved by elimination or reduction of $V_{(k)}$-nodes in the partitioning solution.

## 4.1 $V$-shaped Node Elimination

[2]Of course, such models can be very relevant for, e.g., multi-FPGA partitioning applications that were prominently studied during the early 1990s.

For any node $v_i$, let $I(v_i)$ be the set of nets to which $v_i$ is connected that lie entirely in the current partition of $v_i$, and let $O(v_i)$ be the set of nets that belong to the cutset and for which $v_i$ is the only incident node in the partition of $v_i$. The traditional gain function in FM partitioning is: $g(v_i) = |O(v_i)| - |I(v_i)|$ for all nodes $v_i$. FM partitioning [10] starts with a random initial partition and iteratively checks the node with maximum gain to see whether moving it to the other part will violate the area balance constraint. If not, the node is moved to the other part, otherwise the node with maximum gain in the other subset will be moved. Every node is locked after moving, and the process continues until all nodes are locked. Then, all prefix sums $S_t = \sum_{j=1}^{t} g(v_j)$ are calculated, and $q$ is chosen such that $S_q$ is maximum (all node moves after the $q^{th}$ are undone). This process is called a *pass*, and FM partitioning repeats passes until $S_q \leq 0$.

In general, the partitioning solutions returned by a multilevel FM partitioner, such as MLPart [4], have good cutsize. However, for some testcases, MLPart tends to produce solutions with high $h$ values. We have analyzed critical paths and consistently found that:

1. there are a few $V$-shaped nodes in the partitioning solutions;
2. every $V$-shaped node is included in many critical paths;
3. almost every critical path contains one or more $V$-shaped node; and
4. MLPart cannot eliminate these $V$-shaped nodes due to the traditional gain function.

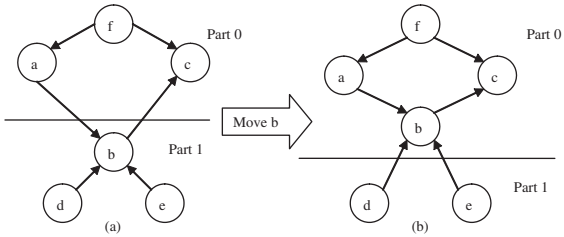


**Figure 1: Example of $V$-shaped node $b$.**

Based on these observations, we propose to improve timing by local biasing of FM to eliminate some (not necessarily all) $V$-shaped nodes. For example, in Figure 1(a), node $b$ is a $V$-shaped node. Suppose that the area balance tolerance is 0.35. MLPart cannot move node $b$ to Part 0 since there are directed edges from nodes $d$ and $e$ to $b$. The traditional gain values of nodes $a, b, c, d, e, f$ are 0, 0, 0, -1, -1, -1. Since the smallest cutsize is already achieved and no improvement is available, the FM partitioner will stop here. Of the directed paths passing through $b$, $..a \rightarrow b \rightarrow c..$ will have two cut hyperedges; $..d \rightarrow b \rightarrow c..$ and $..e \rightarrow b \rightarrow c..$ will each have one cut hyperedge. However, if we move node $b$ to Part 0 as shown in Figure 1(b), although the cutsize remains the same, the two cut hyperedges on the path $..a \rightarrow b \rightarrow c..$ are saved while the number of cut hyperedges on the other two paths remains at one. We see that unlike

timing-analysis based algorithms, which may increase the hopcounts of near-critical paths when the hopcounts of critical paths are reduced, elimination of $V$-shaped nodes can improve timing without any negative effect.

We believe that this effect is increasingly important in recent industry testcases: partitioning solutions have much worse timing if they do not consider $V$-shaped nodes. However, this effect is not apparent for testcases in which most gates have only two inputs. For example, in Figure 1(a), if node $e$ is removed, the gain value of node $b$ will be 1, and the FM partitioner will move node $b$ to Part 0. For such testcases, there will be very few $V$-shaped nodes in the ML-Part partitioning solution. Our method may therefore be more suited to the "true" underlying netlist topology after synthesis, and in fact is not effective if the netlist has been reduced to some sort of generic 2-input gate variant.[3]

## 4.2 Elimination of Generalized $V$-shaped Nodes

For some testcases, eliminating $V$-shaped nodes is not sufficient since there are many $V_{(2)}$-nodes and $V_{(3)}$-nodes left in the partition solutions after elimination of $V$-shaped nodes. Moving these nodes can make the solutions better. Ideally, we hope that no two cut hyperedges with different directions (one $i$ and one $o$ cut hyperedge) are located too close to each other in a path.[4] Therefore, we want to eliminate $V_{(k)}$-nodes, with $k$ as large as possible. However, the number of $V_{(k)}$-nodes will increase dramatically with $k$, which means that we likely need to change the pure mincut-driven solution more substantially with large $k$. Another problem associated with large $k$ is that movement of one node may affect many other nodes, again leading to increased cutsize and complexity. Currently, we do not believe that it is practical to be concerned with $k \geq 4$.

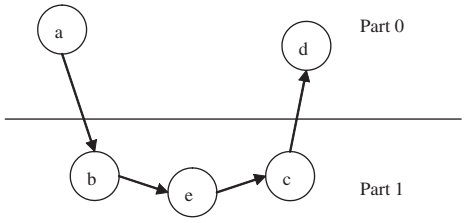


**Figure 2: Example of $V_{(3)}$-node.**

To eliminate $V_{(k)}$-nodes, we effectively need to move more than one node. For example, in Figure 2, we need to move nodes $b$, $c$, $e$ in order to eliminate $V_{(3)}$-node $b$. If just node $b$ is moved, $e$ will become a new $V_{(2)}$-node. For any $V_{(k)}$-node $v_i$, there is a set of nodes whose movement is required to eliminate $v_i$. We denote this set as $MS(v_i)$. In Figure 2, $MS(b) = \{b, c, e\}$. Therefore, after moving one node $v_i$, we need to use a "look ahead" algorithm to in effect move the rest of the nodes in $MS(v_i)$; we achieve this by means similar to the CLIP algorithm of Dutt et al. [9].

[3]Note that ISCAS sequential benchmarks have been broken down into 2-input gates, and thus are not an interesting context for us.

[4]The intuition is as follows. For a FF-FF path $P$ of 50 nodes, the max possible hopcount is 49. If we require that the distance between any two opposite-direction cuts is no smaller than 5, the max possible hopcount is 9. By making $k$ large enough, we can force the decrease of $h((V_0, V_1)) = 1$ to its minimum value.

**Procedure to Calculate $r_j(v_i)$**

1. **If** $((v_i) \in V_c)$ and $(In(v_i) \neq \phi))$
2. {
3. Perform BFS from $v_i$ for at most $k$ levels. At each level $j$, delete all sequential nodes and store the remaining nodes before going to the next level.
4. **For** $(j = 1;\ j \leq k;\ j++)$
5. {
6. **If** $((\exists$ one node $\in V_1$ at level $j)$ and $(In(v_i) \bigcap V_1 \neq \phi))$
7. $r_j^{'}(v_i) = 1$
8. **Else** $r_j^{'}(v_i) = 0$
9. **If** $((\exists$ one node $\in V_0$ at level $j)$ and $(In(v_i) \bigcap V_0 \neq \phi))$
10. $r_j^{''}(v_i) = 1$
11. **Else** $r_j^{''}(v_i) = 0$
12. $r_j(v_i) = r_j^{'}(v_i) - r_j^{''}(v_i)$
13. }
14. }

**Figure 3: Procedure to Calculate $r_j(v_i)$.**

## 4.3 New Gain Function Calculation

To achieve what we call "distance-$k$ unidirectional bias", we change the gain function to:

$$Gain(v_i) = \delta(0)g(v_i) + \sum_{j=1}^{k} \delta(j) r_j(v_i) \text{ for each node } v_i$$

Here, $g(v_i)$ is the traditional net-cut based gain function. The user-defined coefficients $\delta(j) \geq 0$ weight the attention paid to different $V$-shapes (if only $\delta(0)$ is positive, then we have the original FM algorithm), and $r_j(v_i)$ is the reduction in the number of $V_{(j)}$-nodes after moving $v_i$. For any node $v_i \in V_c$, we denote the set of inputs of $v_i$ as $In(v_i)$. To simplify the description, we assume that $v_i \in V_0$. The procedure to calculate $r_j(v_i)$ is shown in Figure 3.

In Steps 1-3 of the procedure, if $v_i$ is a combinational node and its input set is not empty, we find all the nodes within distance $k$ of $v_i$ using BFS. All sequential nodes and their descendants will be removed from the BFS tree. Then for every level $j$ from 1 to $k$, we check whether $v_i$ is a $V_{(j)}$ node in Steps 6-8. If so, $r_j^{'}(v_i) = 1$. In Steps 9-11, we check whether $v_i$ will be a $V_{(j)}$ node if it is moved to the other part. The value of $r_j(v_i)$ can be 1, 0, -1, which represents the reduction of the number of $V_{(j)}$ nodes in the hypergraph due to the move of $v_i$.

To analyze the time complexity of the procedure, assume that the maximum fanin is $MI$ and the maximum fanout is $MO$. For every node, checking the inputs will take at most $O(MI)$ time and BFS will take at most $O(MO^k)$ time. Therefore, the total time needed for calculating $r_j(v_i)$ is $O(MO^k + MI)$.

## Distance-$k$ Unidirectional Bipartitioning: Summary and Time Complexity

Our algorithm to achieve distance-$k$ unidirectional bias in the bipartitioning (reminiscent of CLIP [9] in how it induces movement of clusters across the cutline) is summarized in Figure 4. Time complexity may be analyzed as follows. We use the same gain bucket list structure as proposed in [10]. The maximum possible gain is $Gain_{max} = \delta(0)g_{max} + \sum_{j=1}^{k} \delta(j)$, where $g_{max}$ is the maximum possible

**Distance-$k$ Unidirectional Bipartitioning Algorithm**

1. Calculate initial gains for all nodes and store the gains into two gain buckets $B_0$ and $B_1$.
2. Select the node $v_i$ with the maximum gain.
3. Reset the gains of all nodes to zero.
4. Move $v_i$ and update the gains of $v_i$ and its neighbors.
5. **While** ($\exists$ one node that has not been moved)
6. {
7. Select one node $v_j$ with maximum updated gain whose movement satisfies the area balance constraint.
8. Move $v_j$ and update the gains of $v_j$ and its neighbors.
9. Lock $v_j$ and record its gain.
10. }
11. Find the point in the move sequence at which the sum of gains is maximum. Undo all the moves after this point.

**Figure 4: Bipartitioning Algorithm.**

traditional gain. The time for calculating initial gain is $O((MO^k + MI)n)$, where $n$ is the number of nodes in the hypergraph. $O(Gain_{max})$ time is needed to reset the gains of all nodes to zero, since we only need to remove all linked lists from buckets and concatenate them to the bucket of zero gain. Moving gains also takes $O(Gain_{max})$ time. Since moving one node only affects the gains of the nodes within distance $k$, we need to update at most $O(MO^k + MI^k)$ nodes in every iteration. Because every node can be moved at most once, the total time should be $O((MO^k + MI^k + Gain_{max})n)$. Therefore, our algorithm takes linear time per pass, just as in the original FM algorithm. The negligible impact on runtime is confirmed in the next section.

## 5. Experimental Results

The MLPart code of [4] was downloaded from the MARCO GSRC Bookshelf [14] and modified. The code is currently compiled and run on Solaris and Linux platforms. Total code modifications amounted to less than 2,000 lines.

We tested our algorithm on four industry testcases given to us in LEF/DEF format. The testcase parameters are summarized in Table 1. All tests were run on code compiled with the GNU gcc2.95.2 compiler running on a 600MHz Intel Pentium-III Xeon processor under the RedHat7.3 Linux operating system. Runtimes are reported in seconds. We use the model in [2] to calculate the delay.

| Testcase | #Nets | #Nodes | #FF | #PI | #PO |
|---|---|---|---|---|---|
| industry1 | 16377 | 15768 | 729 | 599 | 1 |
| industry2 | 21947 | 21219 | 3630 | 1238 | 256 |
| industry3 | 32699 | 28546 | 16242 | 815 | 628 |
| industry4 | 21200 | 21397 | 2713 | 164 | 175 |

**Table 1: Basic Properties of testcases.**

Table 2 shows the results of multiple single-start runs on the testcase "industry1" when run with $\delta(0) = 1$, $\delta(1) = 10$. The results show that around 30% improvement on hop-count and 23% improvement on delay is achieved while only slightly increasing cutsize and runtime.[5]

We also tested our algorithm with different values of $\delta(0)$, $\delta(1)$, and $\delta(2)$ for the testcase "industry1" in order to find the best tuning of parameter values. The results are shown in Table 3. In each test, we choose a different combination of values from the set {1, 3, 10, 30} for each of the three

[5]When using the simpler delay model of [7], we obtain 20% improvement in delay.

| Trial | MLPart | | | | ML+V-nodes Removal | | | |
|---|---|---|---|---|---|---|---|---|
| | cutsize | h | delay | CPU(s) | cutsize | h | delay | CPU(s) |
| 1 | 803 | 5 | 330.4 | 11.97 | 832 | 3 | 262.1 | 12.14 |
| 2 | 821 | 6 | 378.5 | 11.88 | 853 | 4 | 277.6 | 12.45 |
| 3 | 815 | 5 | 330.4 | 12.03 | 844 | 3 | 262.1 | 12.38 |
| 4 | 817 | 6 | 378.5 | 11.65 | 848 | 4 | 277.6 | 12.67 |
| 5 | 839 | 5 | 330.4 | 11.56 | 844 | 3 | 262.1 | 13.01 |
| 6 | 822 | 5 | 330.4 | 11.63 | 852 | 3 | 262.1 | 12.91 |
| 7 | 823 | 5 | 330.4 | 11.89 | 852 | 3 | 262.1 | 12.32 |
| 8 | 841 | 6 | 378.5 | 11.64 | 863 | 4 | 277.6 | 12.31 |
| 9 | 798 | 5 | 330.4 | 11.71 | 824 | 3 | 262.1 | 12.42 |
| 10 | 828 | 5 | 330.4 | 11.92 | 847 | 3 | 262.1 | 12.57 |

**Table 2: Biasing against $V$-shaped nodes versus ML-Part: detailed results of 10 independent single-start trials for the testcase industry1. The parameters used in our algorithm are $\delta(0) = 1$ and $\delta(1) = 10$; area balance tolerance is 10%.**

| $\delta$ | | | cutsize | | | $h$ | | | delay | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | Min | Max | Avg | Min | Max | Avg | Min | Max | Avg |
| 1 | 1 | 1 | 823 | 841 | 833.7 | 3 | 4 | 3.2 | 262.1 | 277.6 | 272.3 |
| 1 | 1 | 3 | 837 | 857 | 848.5 | 3 | 4 | 3.2 | 262.1 | 296.2 | 285.3 |
| 1 | 1 | 10 | 841 | 873 | 858.5 | 3 | 4 | 3.3 | 262.1 | 296.2 | 288.5 |
| 1 | 1 | 30 | 851 | 878 | 867.9 | 3 | 4 | 3.3 | 262.1 | 287.7 | 285.3 |
| 1 | 3 | 1 | 837 | 869 | 852.1 | 3 | 4 | 3.1 | 262.1 | 287.7 | 270.1 |
| 1 | 3 | 3 | 841 | 867 | 854.8 | 3 | 4 | 3.2 | 262.1 | 262.1 | 262.1 |
| 1 | 3 | 10 | 841 | 864 | 851.3 | 3 | 4 | 3.1 | 262.1 | 277.6 | 270.1 |
| 1 | 3 | 30 | 831 | 848 | 839.7 | 3 | 4 | 3.2 | 262.1 | 287.7 | 268.6 |
| 1 | 10 | 1 | 829 | 866 | 845.1 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 1 | 10 | 3 | 831 | 872 | 851.1 | 3 | 4 | 3.1 | 262.1 | 277.6 | 270.1 |
| 1 | 10 | 10 | 844 | 866 | 855.9 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 1 | 10 | 30 | 846 | 866 | 852.4 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 1 | 30 | 1 | 830 | 855 | 840.7 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 1 | 30 | 3 | 835 | 867 | 847.5 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 1 | 30 | 10 | 856 | 876 | 866.4 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 1 | 30 | 30 | 842 | 869 | 859.5 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 3 | 1 | 1 | 833 | 850 | 844.1 | 3 | 4 | 3.2 | 262.1 | 296.2 | 280.9 |
| 3 | 1 | 3 | 821 | 852 | 840.7 | 3 | 4 | 3.4 | 262.1 | 296.2 | 284.4 |
| 3 | 1 | 10 | 840 | 869 | 860.1 | 3 | 4 | 3.4 | 262.1 | 296.2 | 283.2 |
| 3 | 1 | 30 | 850 | 872 | 857.3 | 3 | 4 | 3.3 | 262.1 | 296.2 | 283.2 |
| 3 | 3 | 1 | 827 | 843 | 835.7 | 3 | 4 | 3.3 | 262.1 | 277.6 | 268.6 |
| 3 | 3 | 10 | 837 | 857 | 848.5 | 3 | 4 | 3.2 | 262.1 | 296.2 | 280.9 |
| 3 | 10 | 1 | 843 | 857 | 851.5 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 3 | 10 | 3 | 839 | 868 | 852.4 | 3 | 4 | 3.2 | 262.1 | 287.7 | 266.7 |
| 3 | 10 | 10 | 844 | 869 | 855.1 | 3 | 4 | 3.2 | 262.1 | 262.1 | 262.1 |
| 3 | 10 | 30 | 840 | 863 | 851.1 | 3 | 4 | 3.1 | 262.1 | 277.6 | 270.1 |
| 3 | 30 | 1 | 828 | 847 | 835.4 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 3 | 30 | 10 | 842 | 861 | 853.4 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 10 | 1 | 1 | 821 | 852 | 840.7 | 3 | 4 | 3.4 | 262.1 | 296.2 | 284.4 |
| 10 | 1 | 3 | 824 | 847 | 842.6 | 3 | 4 | 3.6 | 262.1 | 296.2 | 285.7 |
| 10 | 1 | 10 | 832 | 868 | 854.2 | 3 | 4 | 3.4 | 262.1 | 296.2 | 284.4 |
| 10 | 1 | 30 | 846 | 873 | 857.1 | 3 | 4 | 3.4 | 262.1 | 296.2 | 286.1 |
| 10 | 3 | 1 | 823 | 848 | 837.3 | 3 | 4 | 3.5 | 262.1 | 296.2 | 286.1 |
| 10 | 3 | 3 | 833 | 852 | 844.3 | 3 | 4 | 3.2 | 262.1 | 296.2 | 280.9 |
| 10 | 3 | 10 | 824 | 856 | 844.7 | 3 | 4 | 3.4 | 262.1 | 296.2 | 284.4 |
| 10 | 3 | 30 | 837 | 867 | 858.9 | 3 | 4 | 3.4 | 262.1 | 296.2 | 285.7 |
| 10 | 10 | 1 | 825 | 849 | 843.7 | 3 | 4 | 3.1 | 262.1 | 262.1 | 262.1 |
| 10 | 10 | 3 | 827 | 845 | 836.3 | 3 | 4 | 3.3 | 262.1 | 277.6 | 268.6 |
| 10 | 30 | 1 | 841 | 868 | 851.7 | 3 | 4 | 3.2 | 262.1 | 287.7 | 266.7 |
| 10 | 30 | 3 | 842 | 854 | 850.6 | 3 | 3 | 3 | 262.1 | 262.1 | 262.1 |
| 30 | 1 | 1 | 824 | 857 | 836.8 | 4 | 5 | 3.7 | 296.2 | 330.4 | 322.4 |
| 30 | 1 | 3 | 820 | 849 | 832.3 | 3 | 5 | 4.2 | 262.1 | 330.4 | 289.1 |
| 30 | 1 | 10 | 826 | 861 | 846.6 | 3 | 4 | 3.2 | 262.1 | 296.2 | 286.1 |
| 30 | 1 | 30 | 838 | 864 | 854.9 | 3 | 4 | 3.4 | 262.1 | 296.2 | 287.3 |
| 30 | 3 | 1 | 814 | 842 | 825.7 | 4 | 5 | 4.3 | 262.1 | 330.4 | 289.1 |
| 30 | 3 | 10 | 826 | 849 | 842.4 | 3 | 4 | 3.5 | 262.1 | 296.2 | 287.3 |
| 30 | 10 | 1 | 831 | 856 | 847.4 | 3 | 4 | 3.1 | 262.1 | 262.1 | 262.1 |
| 30 | 10 | 3 | 826 | 851 | 838.4 | 3 | 4 | 3.5 | 262.1 | 296.2 | 286.8 |
| 30 | 30 | 1 | 824 | 843 | 834.9 | 3 | 4 | 3.2 | 262.1 | 277.6 | 266.7 |

**Table 3: Results for different values of $\delta(0)$, $\delta(1)$, and $\delta(2)$ for the testcase industry1. Equivalents such as (3, 3, 3) and (10, 10, 10) give identical results and are deleted from the table.**

| Testcase | MLPart | | | | Reweighting | | | | MLPart+$V_{(2)}$- nodes Removal | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | cutsize | h | delay | CPU(s) | cutsize | h | delay | CPU(s) | cutsize | h | delay | CPU(s) |
| industry1 | 820.7 | 5.3 | 352.8 | 11.79 | 851.2 | 4.2 | 326.4 | 110.5 | 847.5 | 3 | 262.1 | 13.16 |
| industry2 | 169.9 | 3.5 | 220.7 | 13.45 | 191.4 | 2.7 | 214.6 | 102.3 | 183.2 | 2 | 202.5 | 15.67 |
| industry3 | 141.3 | 3 | 291.6 | 16.67 | 152.3 | 2.6 | 288.4 | 140.7 | 149.2 | 2 | 275.6 | 18.92 |
| industry4 | 408.7 | 5.3 | 302.6 | 12.43 | 466.8 | 4.7 | 288.1 | 112.6 | 416.7 | 3.4 | 243.5 | 14.79 |

**Table 4: Biasing against $V_{(k)}$-nodes (up to $k = 2$) versus MLPart and Reweighting: average results of 10 random starts. The parameters used in our method are $\delta(0) = 1$, $\delta(1) = 30$, $\delta(2) = 3$.**

parameters, and run the code with 10 independent random starts. The minimum, maximum and average of the ten results are reported. The table shows that even (30 1 1) (with very little weight on the $\delta(1)$ and $\delta(2)$ components) achieves very strong improvements in hopcount (average of 39% reduction) with very little change in cutsize (average of 2% increase). Empirically, we believe that good results are consistently achieved with $\delta(0) = 1, \delta(1) = 30$, and $\delta(2) = 3$.

Table 4 gives the average results of all the four testcases with ten random starts, comparing directly against MLPart [4] and Reweighting.[6] We set $\delta(0) = 1,\ \delta(1) = 30,\ \delta(2) = 3$. The results show that our algorithm is very efficient in reducing hopcount as well as delay. The increase of cutsize and runtime is trivial (less than a few percent).

## 6. Conclusions

In this paper, we have proposed a simple yet efficient timing-driven partitioning algorithm which does not rely on any global timing analysis. Since only local information is used in the algorithm, we achieve an effective return of solution quality versus runtime. By changing the gain function in the FM partitioner, we bias toward movement of some $V_{(k)}$-nodes in the FM partitioning solution across the cutline. We have observed that these "bad nodes", that is, $V_{(k)}$-nodes, contribute significantly to the delay of the whole circuit; thus, our biasing approach improves timing by eliminating or minimizing such nodes. Experimental results show that our method significantly reduces path delay while keeping the cutsize and runtime almost the same as MLPart.

Future research includes studying the impact of the new partitioner within the framework of top-down, partitioning-based, timing driven placement. Note that after a length-$l$ path has been placed, its total "hopcount" is basically equal to $l$. However, by reducing the hopcount at the upper levels of the partitioning hierarchy, we may be able to reduce the number of long edges in the placement of the timing path.[7] We also seek efficient ways of biasing the partitioning result for $\delta(k)$, $k > 2$.

[6]Programs described in [7] and [2] were not available for comparison. The reweighting code is obtained by modifying MLPart [14] according to the algorithm proposed in [2].

[7]Suppose that at each level $i$, one hop will contribute to the placed pathlength by $l_i$ on average. Our algorithm can achieve $\delta(h_i)$ reduction in the number of hops, versus traditional multilevel FM partitioners. We expect that we can reduce the total path length by $\sum_{i=1}^{t} \delta(h_i) l_i$ over the first $t$ levels of the partitioning hierarchy (for small values of $t$); these are the hops that have greatest expected edge length in top-down placement.

## 7. REFERENCES


[1] C. Ababei and K. Bazargan, "Statistical Timing Driven Partitioning for VLSI Circuits", *Proc. Design Automation and Test in Europe*, 2002, pp. 1109.

[2] C. Ababei, S. Navaratnasothie, K. Bazargan and G. Karypis, "Multi-objective Circuit Partitioning for Cutsize and Path-Based Delay Minimization", *Proc. IEEE-ACM Intl. Conf. on Computer-Aided Design*, 2002, pp. 181-185.

[3] A.E. Caldwell, A. B. Kahng and I. L. Markov, "Hypergraph Partitioning for VLSI CAD: Method for Heuristic Development, Experimentation and Reporting", *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 349-354.

[4] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartition", *Proc. Asia and South Pacific Design Automation Conf.*, 2000, pp. 661-666.

[5] J. Cong, S. Lim and C. Wu, "Performance Driven Multi-level and Multiway Partitioning with Retiming", *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 274-279.

[6] J. Cong, H. Li and C. Wu, "Simultaneous Circuit Partitioning/Clustering with Retiming", *Proc. ACM/IEEE Design Automation Conf.*, 1999, pp. 460-465.

[7] J. Cong and C. Wu, "Global Clustering-Based Performance Driven Circuit Partitioning", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2002, pp. 149-154.

[8] W. E. Donath, R. J. Norman, B. K. Agrawal, S. E. Bello, S. Y. Han, J. M. Kurtzberg, P. Lowy and R. I. McMillan, "Timing Driven Placement Using Complete Path Delays", *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 84-89.

[9] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 194-200.

[10] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", *Proc. Nineteenth Design Automation Conf.*, 1982, pp. 175-181.

[11] Y. C. Ju and R.A. Saleh, "Incremental Techniques for the Identification of Statically Sensitizable Critical Paths", *Proc. ACM/IEEE Design Automation Conf.*, 1991, pp. 541-546.

[12] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Domain", *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526-529.

[13] L. Liu, M. Shih, N. Chou, C-K. Cheng and W. Ku, "Performance-Driven Partitioning Using a Replication Graph Approach", *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 206-210.

[14] http://nexus6.cs.ucla.edu/GSRC/bookshelf/Slots/Partitioning/MLPart/.