

DETECTING TANGLED LOGIC STRUCTURES IN VLSI NETLISTS

A Thesis

by

TANUJ JINDAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Computer Engineering

DETECTING TANGLED LOGIC STRUCTURES IN VLSI NETLISTS

A Thesis

by

TANUJ JINDAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Jiang Hu
Committee Members,	Peng Li
	Anxiao Jiang
Head of Department,	Costas N. Georghiades

August 2010

Major Subject: Computer Engineering

ABSTRACT

Detecting Tangled Logic Structures in VLSI Netlists. (August 2010)

Tanuj Jindal, B.Tech., Indian Institute of Technology-Roorkee, India

Chair of Advisory Committee: Dr. Jiang Hu

This thesis proposes a new problem of identifying large and tangled logic structures in a synthesized netlist. Large groups of cells that are highly interconnected to each other can often create potential routing hotspots that require special placement constraints. They can also indicate problematic clumps of logic that either require resynthesis to reduce wiring demand or specialized datapath placement. At a glance, this formulation appears similar to conventional circuit clustering, but there are two important distinctions. First, we are interested in finding large groups of cells that represent entire logic structures like adders and decoders, as opposed to clusters with only a handful of cells. Second, we seek to pull out only the structures of interest, instead of assigning every cell to a cluster to reduce problem complexity. This work proposes new metrics for detecting structures based on Rent's rule that, unlike traditional cluster metrics, are able to fairly differentiate between large and small groups of cells. Next, we demonstrate how these metrics can be applied to identify structures in a netlist. Finally, our experiments demonstrate the ability to predict and alleviate routing hotspots on a real industry design using our metrics and method.

To my family

ACKNOWLEDGMENTS

I am very grateful to my advisor Dr. Jiang Hu for giving me this opportunity to work under him. Without his constant guidance, suggestions and encouragement, this work would not have been possible. I owe him my gratitude for showing me this way of research. He has supported and encouraged me whenever I needed him and answered all my questions very openly. The informal talks that I had with him have been a constant source of knowledge and inspiration. I also want to thank him for all the facilities and support he has given to me. Thanks a lot Jiang for everything.

I would also like to thank Chuck Alpert, Zhuo Li, Gi-Joon Nam from IBM Austin Research Laboratory for making me work on such an interesting topic. I am also grateful to them for having constant reviews on my progress and providing constructive feedback. My special thanks goes to Zhuo Li and Gi-Joon Nam for helping run experiments on industrial design.

I would like to thank Dr. Peng Li and Dr. Anxiao Jiang for agreeing to be my committee members. I would also like to thank Dr. Paul Gratz for substituting for Dr. Peng Li in my thesis presentation.

I would also like to thank my parents for the confidence they had in me, without which I would have never been able to pursue and complete my masters study.

Finally, I would like to thank Texas A&M University and the Department of Electrical and Computer Engineering for giving me the opportunity to pursue my masters degree.

TABLE OF CONTENTS

CHAPTER		Page
I	INTRODUCTION	1
II	PREVIOUS WORK	4
III	OUR APPROACH	7
	3.1. Metrics for Tangled Logic Structures	7
	3.2. A Method to Find Groups of Tangled-Logic	10
	3.2.1. Phase I: Linear Ordering Generation	11
	3.2.2. Phase II: Initial Candidate GTL Generation	12
	3.2.3. Phase III: GTL Refinement and Pruning	12
IV	ALGORITHM	14
	4.1. Algorithm Analysis	14
	4.1.1. Algorithm	14
	4.1.2. Complexity Analysis	16
V	EXPERIMENTS	18
	5.1. Experimental Results	18
	5.1.1. Experiments on Random Graphs	18
	5.1.2. Experiments on ISPD Benchmarks	19
	5.1.3. Experiments on an Industrial Circuit	20
VI	CONCLUSIONS	25
	REFERENCES	26
	VITA	27

LIST OF TABLES

TABLE		Page
1	Experimental results on random graphs.	18
2	Experimental results on ISPD 05/06 placement benchmarks.	22
3	GTLs found on the industrial circuit.	22

LIST OF FIGURES

FIGURE		Page
1	Example of routing hotspots.	2
2	Example of nGTL-Score.	8
3	Example of density-aware GTL-Score.	10
4	GTL found by our method in Bigblue1.	20
5	Functions of nGTL-Score (nGTL-S), density-aware GTL-Score (GTL-SD) and ratio cut $\frac{T(C)}{ C }$ versus groups extracted from a linear ordering of cells from Bigblue1.	21
6	GTL of the industrial circuit.	23
7	Routing congestion after cell inflation using GTLs information.	23

CHAPTER I

INTRODUCTION

During logic synthesis, high-level logic structures are translated into groups of logic gates. This synthesized netlist is then handed off to a place-and-route physical design flow. During this handoff, information about the origin of the logic that created the gates can be lost, especially if one switches from one tool vendor to another. Most of the placement literature and all academic placers (e.g., [1] [2]) also assume that logic information is absent and operate purely at the gate level, instead of relying on hierarchical information.

During this handoff between synthesis and placement, logic may be synthesized in such a way that it might require special care from the placement engine to obtain high quality results. Certain groups of logic will invariably have a higher degree of inter-connectivity than other groups. Let GTL denote a *group of tangled logic*. The automatic detection of GTLs has several potential applications:

- **Routability.** Since a GTL has high interconnectivity, placement engine will naturally want to pull the cells tightly together which often will create a routing hotspot. Figure 1 shows a routing congestion map of a placed industrial design, in which the routing hotspots in the upper part of the design are caused by tangled logic structures that are placed too closely together. Later we show how simple process of cell inflation in a GTL can mitigate such routing congestions.
- **Floorplanning.** Since a GTL will stay together during placement, the designer may wish to form a soft block for the gates in the GTL. Then during placement, the soft block can be translated into placement constraints (like attractions, forces, or move bounds) to drive placement to a higher quality solution.

The journal model is *IEEE Transactions on Automatic Control*.

- **Logic re-synthesis.** Synthesis will typically try to instantiate logic in the most compact form possible, yet this is one of the reasons why logic structures can be so tangled. Prior to placement, a GTL could be resynthesized or re-instantiated to utilize more area, but less interconnect, thereby reducing potential hotspots. Applying this technique to a small fraction of the design will not increase area dramatically.

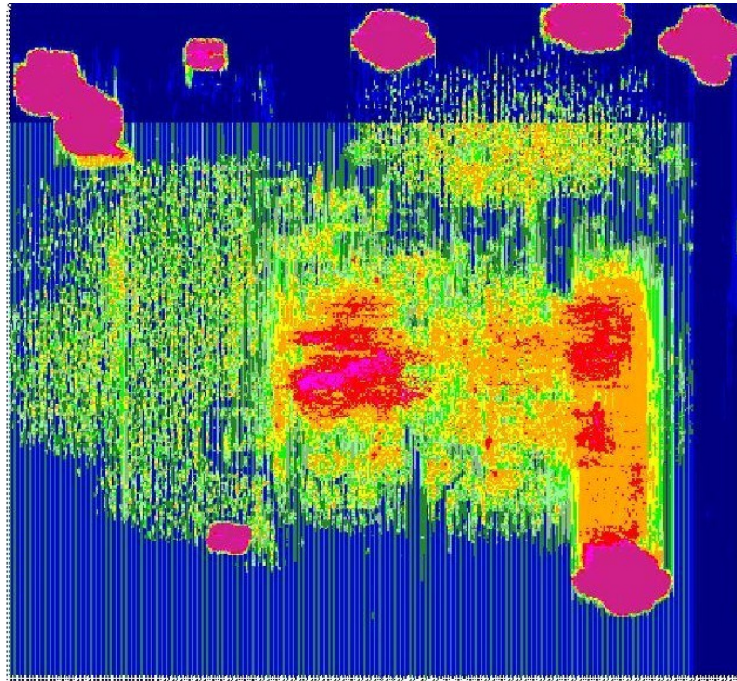


Fig. 1. Example of routing hotspots.

The main problem this work addresses is how to find a GTL. Before one can find a GTL, one should be able to somehow quantify how tangled a logic structure actually is. Therefore, we propose two new metrics derived from Rent's rule to measure the quality of a GTL. The first metric allows one to explore the gamut of sizes between very small and very large cell groups and select the ones which best optimize the metric. Our second metric extends beyond Rent's rule to account for internal connectivity.

The reason a new metric is required is that existing cluster metrics cannot properly

compare groups of cells of different sizes. When searching for GTLs one might find structures within structures, especially as the logic is repeated. We must be able to distinguish between them so that proper guidance can be given to the place-and-route tool. Our metrics and algorithm are able to decide whether we should choose several smaller GTLs or a much larger GTL which encompasses all the smaller ones. Our new metrics are also scaled so that the average score of a typical cluster is one, and the ones with smaller values (e.g., less than 0.1) correspond to strong GTLs. This not only permits one to compare groups of different sizes, it also provides a uniform standard that can be utilized for different designs.

Next we show how one can use the metric to actually find set of GTLs. Our algorithm starts from a random seed and grows a GTL by adding cells iteratively. We exploit parallelism to perform several such searches simultaneously and prune out the GTL candidates that are inferior and overlapping, resulting in an independent final set of identified GTLs.

We validate our metrics and algorithm on random graphs, ISPD placement benchmarks, and a real industrial design. We demonstrate that our algorithm can identify GTL's; and, application of cell inflation technique within each GTL found leads to reduced congestion in our industrial testcase.

CHAPTER II

PREVIOUS WORK

A GTL and a cluster are both a subset of netlist gates, so it might seem that one can apply traditional clustering metrics to GTLs. However, there are some clear differences between the problem statement of detecting tangled logic structures and cell clustering:

1. Conventional clustering most often provides a reduction in problem size. These clusters are typically small (e.g., two to ten cells) so that too much information is not lost in the reduced problem instance. Clustering in this domain is generally local in nature [3]; however, this work is interested in identifying much larger special logic structures, of the order of hundreds to thousands of cells. This requires a more global view that accounts for both external and internal connections.
2. Conventional clustering requires each cell to belong to a cluster, thereby covering the entire netlist. In contrast, we seek specific subsets of cells for special handling prior to placement. Thus, we wish to identify only a small fraction of cells as GTL's and let place-and-route handle the other cells as it wishes.

Let the input netlist be represented as a hypergraph $G = (V, E)$ where V is a set of cells and E is a set of nets, where each $e \in E$ is connected to a subset of V . A clustering is a set of disjoint subsets of cells $C_1, C_2, \dots, C_k \subset V$ such that $V = C_1 \cup C_2 \cup \dots \cup C_k$. Consider the literature of clustering metrics:

1. Given a cluster C , the net cut is defined as the size of the set $T(C) = |\{e \in E | C \cap e \neq \emptyset \& (V - C) \cap e \neq \emptyset\}|$. Clustering metrics can add the cuts in different ways, but fundamentally cut is dependent on cluster size. It is more suited for top-down partitioning or placement, where the sizes of the regions are bounded.

2. *Absorption* [4] is a metric that counts the number of internal connections, and this will grow with cluster size. It is ill-suited for comparing two clusters as possible GTLs since the larger cluster will invariably have larger absorption.
3. The *Ratio Cut* and *Scaled Cost* metrics [5] both treat the cost of a cluster as $\frac{T(C)}{|C|}$. Since $T(C)$ grows much slower than cluster size, a larger cluster will almost always have smaller cost, which makes this a poor way to compare clusters of different sizes.
4. Ng *et al.* [6] proposed using the *Rent exponent* for a cluster as a way of measuring its quality, which means the cost of a cluster C is proportional to $\frac{\ln T(C)}{\ln |C|}$. While this is better than ratio cut, it still monotonically decreases with size as C grows.
5. Hagen *et al.* [7] introduced the concept of DS(Degree Separation) metric. Degree is average number of nets incident on each node in the cluster and Separation is average length of shortest path between any two nodes. They make use of random walk to capture globally good circuit clustering. However, the metric fails to look at the external connection of cluster. Moreover, the authors used the average value of this metric to reflect the overall quality of clustering, not for a single cluster.
6. *(K,L)-connectivity*. In a graph, two nodes are (K,L)-connected if there are K edge-disjoint paths of length no greater than L connecting them. A (K,L)-connected cluster is a subgraph such that every pair of nodes inside are (K,L)-connected. In general, a cluster is highly-connected if it is (K,L)-connected for a large K and a small L. Garbers, et al., proposed a heuristic [8] to find (K,2)-connected clusters. This approach has two problems when it is applied to find highly-connected logic structures. First, a (K,L)-connected cluster may have large cut size [4]. Second, (K,L)-connectivity is very difficult to estimate and consequently (K,L)-connectivity based algorithms tend to be very slow.

7. *Edge separability* [9]. For an edge between node v_i and v_j , its separability is the min-cut between them. This metric tries to describe a more global view of connections. Like (K,L)-connectivity, the evaluation of edge separability is time consuming. Moreover, edge separability only emphasizes internal connections of a cluster while neglecting external connections.
8. *Adhesion* [10]. Adhesion is defined as the sum of min-cuts of all pairs of nodes in a cluster. This metric requires expensive computation cost and is hardly practical for designs with millions of cells.

In summary, none of the clustering literature compares clusters of different sizes without biasing towards either smaller or bigger clusters. Our metrics are the first to do so.

CHAPTER III

OUR APPROACH

3.1. Metrics for Tangled Logic Structures

Our metrics are motivated by the need to (i) compare clusters of different sizes and (ii) measure the tangledness of the group of cells. We start with the ratio cut RC and Rent metric $Rent$ for each cluster C , as discussed previously:

$$RC(C) = \frac{T(C)}{|C|} \quad Rent(C) \propto \frac{\ln T(C)}{\ln |C|}.$$

The problem with both metrics is that the numerator (related to cut) and the denominator (related to cluster size) do not scale together. However, from Rent's rule, we know that $T(C)$ should grow proportionally to $|C|^p$, where p is the Rent exponent. Thus, we define the *GTL-Score* as:

$$GTL-S(C) = \frac{T(C)}{|C|^p}$$

In general, one would expect this metric to be constant for an average quality structure. We do not care about tiny clusters with a handful of cells, nor partitions that consume a huge chunk of the circuit.

Let $A(G)$ be the total number of pins in graph G divided by $|V|$, i.e., $A(G)$ is the average pin count of the cell. According to Rent's rule, then $A(G)$ is the expected value of $GTL-S(C)$. Algorithmically, we want to have a rule of thumb about values of our metrics that identifies a good GTL, and this should be comparable across different netlists. Thus, we further refine our metric to the *normalized GTL-Score*:

$$nGTL-S(C) = \frac{T(C)}{A_G \cdot |C|^p}$$

This will cancel out the differences between circuits with many high fanin versus low fanin

gates. With this scaling, the score of an “average quality group” should be one. However, for a GTL, we would expect the value to be significantly smaller.

To illustrate how the metric behaves in practice, consider a cell agglomeration procedure, which picks a random seed cell and then grows the group by iteratively adding highly connected neighbors. We illustrate the procedure through a generated random graph with 250000 cells, in which 40000 cells were made more highly connected internally and less connected externally than the rest of the graph, i.e., the graph had exactly one GTL of size 40000 cells.

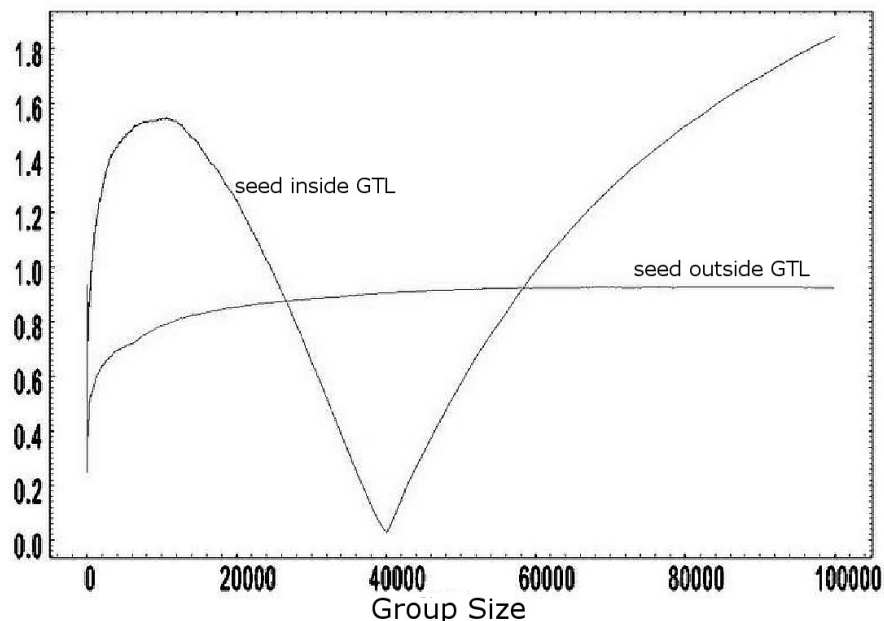


Fig. 2. Example of nGTL-Score.

Figure 2 shows the nGTL-Score as a function of group size for two cell agglomerations. The first was in a set of cells outside the GTL. For this curve, the group starts at a value of 0.3 near group size 0 and then quickly rises and is asymptotically approaching 0.9. However, for the second group inside the GTL, the score rises all the way past 1.5

before dropping precipitously, reaching a local minimum of about 0.1 once the entire GTL was discovered. Adding more cells to the GTL that do not belong causes the score to rise further. The intuition behind this is that, as soon as we include all cells of a GTL in the group $T(C)$ is much smaller than $|C|^p$. And, once we start adding cells from outside the $T(C)$ rises to asymptotically follow $|C|^p$ as proposed by Rent's rule.

So far, we have addressed the issue of comparing groups of different sizes, but the metric does not consider internal connectivity other than what is implied through rent exponent. For a logic structure to be tangled, it should have significantly more internal connectivity versus external connectivity. Often in a design, MUX functions or logic look-up tables are synthesized to a group of complex cells, such as NAND4, OAI, and AOI gates since they generally give the most function per unit area. These gates generally have more pins (four or five) than most of the typical cells, such as AND2/OR2 gates (with three pins). All the connections required for these gates tend to tangle the logic and make the design harder to route. We need to capture the notion of pin-density without disturbing the essence of the normalized GTL metric. We propose to do so as follows:

$$GTL-SD(C) = \frac{T(C)}{A_G \cdot |C|^{p \cdot A_C/A_G}}$$

where A_C is the ratio of the number of pins contained in C divided by $|C|$, i.e., it is the average pin count of cells in the group. The ratio A_C/A_G is close to one when the number of pins inside C is typical relative to the rest of netlist. However, if C contains several complex gates, then this ratio will be higher than one and will reflect stronger likelihood of it being a GTL. Multiplying this value by the Rent exponent biases the cost function to prefer groups of cells with higher pin count and consequently, more tangled logic. This will also provide a check for large cell groups and will identify them as GTL only if they have high density.

Figure 3 shows the same curves as in Figure 2 but with our final *GTL-SD* score.

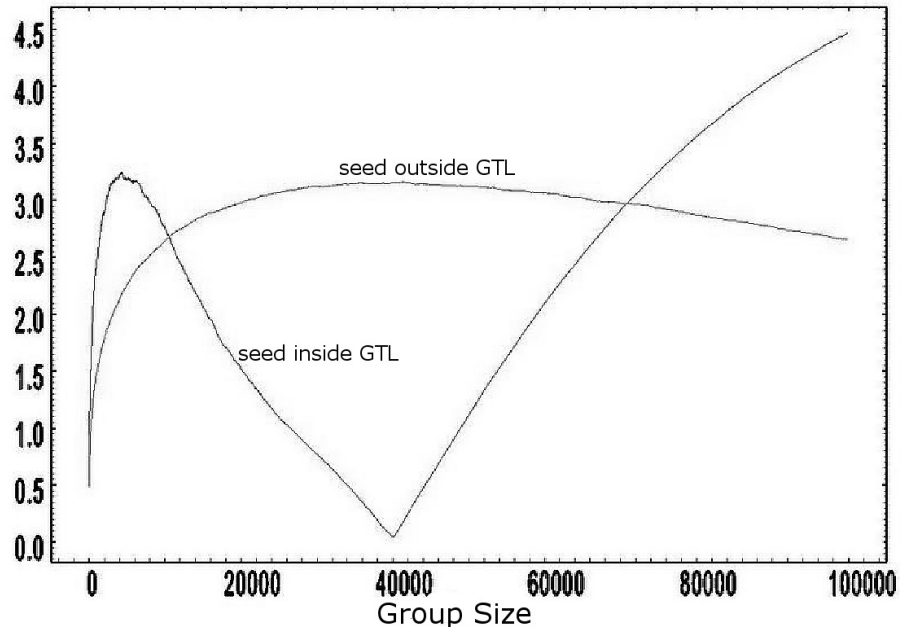


Fig. 3. Example of density-aware GTL-Score.

Comparing the two figures shows that both metrics can reveal the known GTL with 40000 cells. However, the contrast of the local minimum of the *GTL-SD* score is more dramatic than the original metric.

3.2. A Method to Find Groups of Tangled-Logic

Based on our new metrics, we propose a straightforward algorithm (tangled-logic finder) to identify GTLs. This method consists of three phases:

- Phase I: linear ordering generation.
- Phase II: initial candidate GTL generation.
- Phase III: GTL refinement and pruning.

Please note that Phase II and Phase III can be integrated with other linear ordering generation methods [11] as well.

3.2.1. Phase I: Linear Ordering Generation

The linear order generation initializes the group with a seed cell, which is randomly generated. Then, it iteratively adds one cell at a time to the group. The candidates for the cell addition are the cells outside of the group, but with direct edge connections with the group. Among these candidates, we choose the one with the strongest connection with the group. We use a weighted number of nets to indicate the degree of connection. If a candidate cell v_i has a net e connected to the group and this net has $\lambda(e)$ pins outside of the group, its weight is $\frac{1}{\lambda(e)+1}$. Hence, a net has higher weight if it has greater portion of its pins inside the group. The connection between v_i and the group is defined by $\sum_{e|v_i \in e, e \cap C \neq \emptyset} \frac{1}{\lambda(e)+1}$. We use min-cut as a secondary criterion for breaking ties.

In this context, we are simply trying to build groups of connected cells to generate a potential linear ordering. Since the cells are being added iteratively, the cost function is trying to maximize the connectivity.

When selecting among the candidate cells, emphasizing the connection $\sum_{e|v_i \in e, e \cap C \neq \emptyset} \frac{1}{\lambda(e)+1}$ instead of min-cut alone is particularly important at the beginning of cell agglomeration. If a candidate cell is outside the GTL, it usually has weak connections with its neighbors. If we use min-cut as the primary criterion, it is quite likely that this cell is included into the growing group. Likewise, if a candidate cell is inside the GTL, it usually has strong connections with its neighbors and the min-cut criterion may easily exclude this cell. The order in which the cells are added determines the linear ordering. The preference of connection over net-cut leads to building denser groups with low external connectivity. Thereby, leading to addition of cells belonging to true GTL first to the growing group.

3.2.2. Phase II: Initial Candidate GTL Generation

A cell group can be extracted from a linear ordering according to the metrics described in Section 3.1. A group C of size $k = |C|$ is composed by the first k cells in the linear ordering. Then, the function $nGTL-S(C)$ or $GTL-SD(C)$ with respect to k is obtained like in Figure 2 and Figure 3. If there is a clear minimum in this function, the corresponding cell group is selected as a candidate GTL "B". When computing the nGTL-Score, we need to decide the value of Rent exponent p . This is obtained by averaging the Rent exponents for all groups obtained in the linear ordering. The Rent exponent of a group C can be estimated by $\frac{\ln T(C) - \ln A_C}{\ln |C|}$ where A_C is the average number of pins per cell in C .

The procedure described so far is to identify a single GTL in the netlist. If the initial seed is outside of any existing GTLs, this procedure may fail like the flat curves in Figure 2 and Figure 3. To solve this problem, multiple searches starting with different seeds can be performed to generate a population of linear ordering and candidate GTLs $\mathcal{B} = \{B_1, B_2, \dots, B_m\}$ for m parallel runs. If the number of searches is large enough, most of the GTLs can be captured.

3.2.3. Phase III: GTL Refinement and Pruning

A candidate GTL grown from a random seed might be slightly inaccurate. For instance, if the seed is at the boundary of an actual GTL, some cells outside that GTL might be included. In order to solve this problem, we enrich each initial candidate by additional candidate solutions. For each candidate B_i obtained in Phase II, we generate another set of candidates $B_{i,1}, B_{i,2}, \dots, B_{i,l}$ using seeds inside B_i and the same procedure as Phase I and II. These additional candidates are usually close to but slightly different from B_i . Then, union and intersection operations are performed on $\{B_i, B_{i,1}, B_{i,2}, \dots, B_{i,l}\}$ like in genetic algorithm. Finally, the candidate \hat{B}_i with the best score of the proposed metrics is selected

as the refined candidate corresponding to the initial candidate of B_i . This procedure is carried out for all initial candidates in \mathcal{B} to obtain a set of refined candidates $\{\hat{B}_1, \hat{B}_2, \dots, \hat{B}_m\}$. These refined candidates are compared with each other. If one has overlap with another and inferior GTL-Score, it is pruned out. The disjoint candidates remained at the end is the final set of GTLs discovered by our method.

The point to be noted here is that all the three phases mentioned above are computed for all m initial seeds in parallel with no interdependence. The only serial part of algorithm is the final comparison between m refined GTLs generated through parallel execution.

CHAPTER IV

ALGORITHM

4.1. Algorithm Analysis

Below we present a comprehensive view of algorithm used and also complexity associated with it.

4.1.1. Algorithm

Procedure: *TangledLogicFinder*(G, m, Z)

Input: A netlist $G = (V, E)$

Number of seeds m

Maximum linear order length $Z < |V|$

Output: A set of GTLs \mathcal{L}

Phase I: linear ordering generation

- I.1 Randomly generate seeds $s_1, s_2, \dots, s_m \in V$
- I.2 For $i = 1$ to m
- I.3 $S_i \leftarrow s_i, index = 1$
- I.4 $\omega(s) = index$
- I.5 While $|S_i| \leq Z$
- I.6 For all cells neighboring to S_i
- I.7 Find v_j with $\max \sum_{e \in E_j} 1 / (|e| - |e \cap S_i| + 1)$
Break tie by favoring min cut
// E_j is subset of $T(S_i)$ and incident to v_j
- I.8 $index ++$

- I.9 $S_i \leftarrow S_i \cup \{v_j\}$
 I.10 $\omega(v_j) = \text{index}$
 I.11 $\Omega_i = \{\omega(v_j), \forall v_j \in S_i\}$

Phase II: initial candidate GTL generation

- II.1 For $i = 1$ to m
 II.2 Sort cells in S_i in ascending order of ω
 II.3 $C_{i,k} \leftarrow$ first k cells in $S_i, k = 1, 2, \dots, Z$
 II.4 $B_i \leftarrow C_{i,k}$, s.t. $\Phi(C_{i,k})$ is minimized

Phase III: GTL refinement and pruning

- III.1 For $i = 1$ to m
 III.2 Choose 3 random cells from B_i
 III.3 Generate $\{B_{i1}, B_{i2}, B_{i3}\}$ repeating Phase I and
 Phase II for each of the 3 cells
 III.4 $\mathcal{F} \leftarrow \{B_i, B_{i1}, B_{i2}, B_{i3}\}$
 III.5 $T \leftarrow \{B_i, B_{i1}, B_{i2}, B_{i3}\}$
 III.6 For $Z_i \in T$
 III.7 For $Z_j \in \{T - Z_i\}$
 III.8 $\mathcal{F} = \mathcal{F} \cup \{Z_i \cap Z_j\}$
 III.9 $\mathcal{F} = \mathcal{F} \cup \{Z_i \cup Z_j\}$
 III.10 $\mathcal{F} = \mathcal{F} \cup \{Z_i - \{Z_i \cap Z_j\}\}$
 III.11 $\mathcal{F} = \mathcal{F} \cup \{Z_j - \{Z_i \cap Z_j\}\}$
 III.12 $T = \{T - Z_i\}$
 III.13 $\hat{B}_i \leftarrow$ group with minimum Φ in F
 III.14 $\mathcal{C} = \mathcal{C} \cup \hat{B}_i$

- III.15 $\mathcal{L} = \emptyset$
- III.16 Sort \mathcal{C} in non-increasing order of Φ
- III.17 For $i = 1$ to $|\mathcal{C}| - 1$
- III.18 For $j = i + 1$ to $|\mathcal{C}|$
- III.19 If $P_i \cap P_j \neq \emptyset$, break
- III.20 If $j > |\mathcal{C}|$
- III.21 $\mathcal{L} = \mathcal{L} \cup \{P_i\}$
- III.22 Return \mathcal{L}

4.1.2. Complexity Analysis

Complexity analysis is provided for each Phase separately followed by combined analysis of full implementation.

Finding node v_j in Step *I.7* of Phase I takes logarithmic time and is due to removal of v_j from maintained C++ map structure. Taking while loop of *I.5* into account this leads to $O(Z \ln|V|)$ complexity. After finding v_j we have to update the value associated with the neighbors of v_j since only their's net cut and net weight will be affected. In this we ignore nets of v_j with $(|e| - |e \cap S_i|) \geq 20$ as connection weight value associated with cells of these nets is not affected significantly. Let K be a factor to emphasize the fact that we are scanning only a small portion of netlist. Taking the whole loop of *I.5* into account computations upper limit will be $(ZK|E| \ln|V|)$. Hence total complexity for Phase I is $O(|E| \ln|V|)$ and the constant factor associated is also not large.

Phase II consists of finding the GTL from linear ordering of Phase I. This involves finding the minimum and can be achieved in $O(Z)$.

In Phase III (from III.1 to III.3) we first find 3 more groups using seeds inside the GTL found at the end of Phase II. This process involves repeating Phase I and II and hence

total complexity for this part is $O(|E|\ln|V| + Z)$. After this step we perform Union and Intersection between 4 groups so obtained (from II.4 to III.13), each of size Z , having complexity of $O(Z)$.

Here the point to be noted is that all the above procedure can be done in parallel for each initial seed and hence present total complexity of $O(|E|\ln|V| + Z)$ only.

In final pruning stage of Phase III unique GTLs are found among $m \hat{B}$ GTL leading to a complexity of $O(m^2)$.

CHAPTER V

EXPERIMENTS

5.1. Experimental Results

The proposed metrics and methods are tested on various testcases: random graphs, ISPD placement benchmarks [12] and a realistic industrial circuit. The experiments are performed on a Linux server with 8 Intel Xeon processors of 3.2GHz frequency and 8G memory. The algorithm is implemented in C/C++ and parallelized using pthread in 8 parallel threads. In the experiments, the size of each linear ordering is at most 100K cells.

5.1.1. Experiments on Random Graphs

Table 1. Experimental results on random graphs.

Graph Information			Tangled-Logic Finder Solutions						
Case	$ V $	Synthesized GTLs	#seeds	# GTL found	GTL sizes	nGTL-S	GTL-SD	Miss	Over
1	10K	500×1	100	1	501	0.1	0.085	0%	0.2%
2	100K	$2K \times 1 + 15K \times 1$	100	2	2010	0.025	0.022	0%	0.5%
					15003	0.017	0.0156	0.03%	0.05%
3	100K	$5K \times 1$	100	1	5008	0.023	0.043	0%	0.16%
4	800K	$40K \times 6$	100	6	40040	0.0095	0.001	0%	0.1%
					40092	0.0121	0.0209	0.04%	0.27%
					40053	0.0124	0.0214	0.14%	0.28%
					40044	0.0143	0.0015	0%	0.11%
					40044	0.0143	0.0015	0%	0.11%
				40006	0.0191	0.0021	0%	0.02%	

The random graphs are generated based on [8] and its tangled logic structures are known *a priori*. The experimental results on the random graphs are shown in Table 1 and are generated using 100 seeds for each of the run. The second column lists the number of nodes in each graph. The third column describes the synthesized GTLs in the graphs. For example, case 2 has two GTLs: one with 2000 nodes and the other with 15000 nodes. From the fifth column, one can see that our method can find all of the GTLs. In column 6, 7 and 8, the GTL sizes and the values of nGTL-Score (nGTL-S) and density-aware GTL-Score (GTL-SD) are reported. Column 9 tells the percentage of nodes which are in the

known GTL but are missed by our method. Our method has zero missing nodes for 7 of the 10 GTLs. The maximum missing percentage is only 0.14%. Column 10 indicates the percentage of nodes which are not in the GTL but are included by our solution. This rate is also very low and no more than 0.5%. Since our method is to roughly point out the GTLs which need special treatment, missing a few cells or including a few more cells has negligible effect.

5.1.2. Experiments on ISPD Benchmarks

Since for ISPD placement benchmarks we have no knowledge about the existing GTLs in advance, we verify our metrics and method by correlating the solution generated with cell placement results. A placer normally places highly-connected cells close to each other, therefore the cells in a GTL found by our method are expected to be crowded in a small local region. Visualizations of cell placement and our tangled-logic finder solutions is illustrated in Figure 4. The clots with colors different from the majority of cells are the GTLs found by our method. Different color indicates different GTL.

We further compared our metrics with ratio cut [5]. The curves of these metrics versus groups extracted from a linear ordering are shown in Figure 5. The top two curves correspond to the nGTL-Score and the density-aware GTL-Score. The bottom curve is from ratio cut $\frac{T(C)}{|C|}$. The ratio cut curve is much flatter and its global minimum is at its right end. This demonstrates that ratio cut overly favors large group size. Both of the top two curves have global minimum almost at the same place, i.e., they identify the same GTL. The one having the lowest minimum is from the density-aware GTL-Score and the other one is from the nGTL-Score. The curve of nGTL-Score confirms our expectation that the value of nGTL-Score should be mostly around 1.

The experimental results on the ISPD benchmarks are summarized in Table 2. It shows the circuit size $|V|$, number of seeds we used, number of GTLs founded and detailed

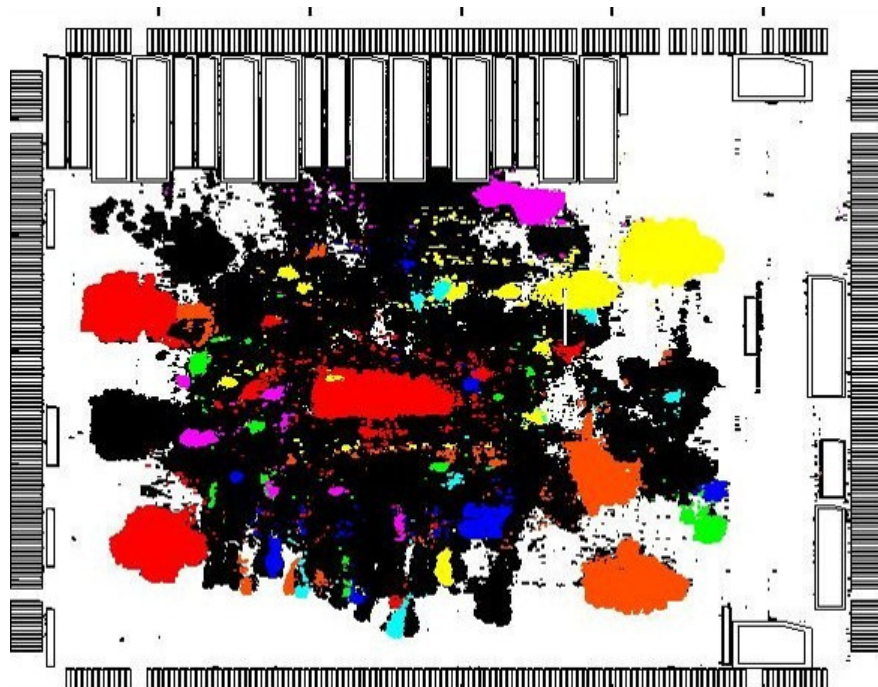


Fig. 4. GTL found by our method in Bigblue1.

information from 3 GTLs of each case. The rightmost column lists the total runtime of our 3-phase method in minutes. One can see that the GTLs can be estimated in 2-3 hours for a million nodes design by using both our method and our metrics. The current run-time obtained is in this range because we are issuing only 8 parallel threads at one time. But in industry, for practical application, we can afford to issue over 100 parallel runs in single step which can reduce the runtime dramatically by a factor of close to 2-5. Moreover, the quoted run-time still has a clear advantage on placement and routing that together takes close to 1 day.

5.1.3. Experiments on an Industrial Circuit

The proposed metrics and methods are also tested on an industrial circuit. Figure 6 displays the tangled-logic finder solutions in cell placement. This is an industrial commercial ASIC

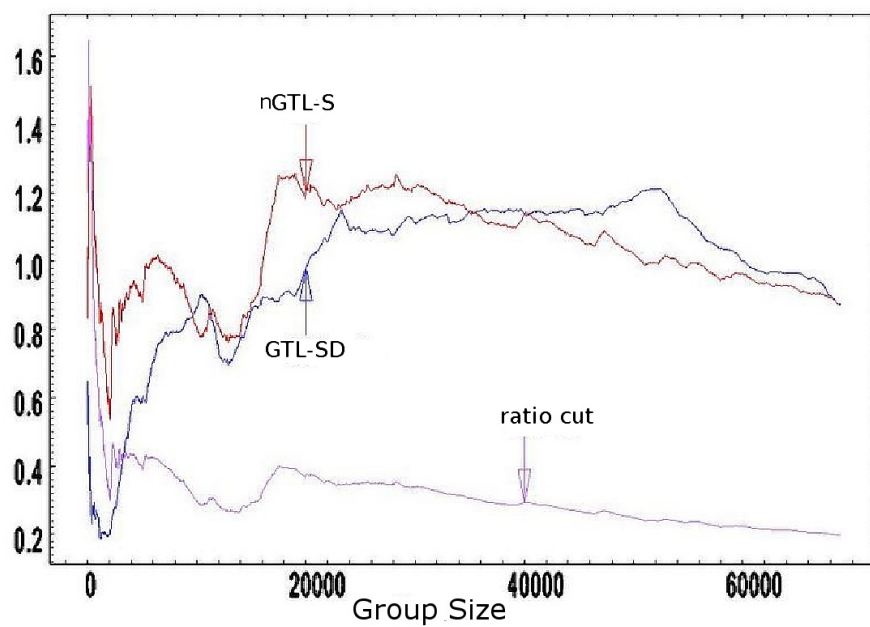


Fig. 5. Functions of nGTL-Score (nGTL-S), density-aware GTL-Score (GTL-SD) and ratio cut $\frac{T(C)}{|C|}$ versus groups extracted from a linear ordering of cells from Bigblue1.

Table 2. Experimental results on ISPD 05/06 placement benchmarks.

Case	$ V $	#seeds	# GTL found	Top 3 GTLs	GTL size	Cut	GTL-S	GTL-SD	Runtime(m)
Bigblue1	278164	100	72	Structure 1	6187	369	0.14	0.031	81
				Structure 2	1548	307	0.32	0.083	
				Structure 3	3539	800	0.46	0.14	
Bigblue2	557786	100	93	Structure 1	13888	397	0.107	0.045	104
				Structure 2	9602	560	0.196	0.111	
				Structure 3	10776	1091	0.352	0.195	
Bigblue3	1096812	100	112	Structure 1	695	81	0.204	0.225	159
				Structure 2	297	76	0.354	0.202	
				Structure 3	13005	2289	0.686	0.454	
Adaptecl	211447	100	78	Structure 1	2628	124	0.128	0.083	77
				Structure 2	2616	136	0.141	0.093	
				Structure 3	375	36	0.142	0.212	
Adaptecl2	255023	100	54	Structure 1	751	52	0.132	0.315	114
				Structure 2	3387	263	0.236	0.058	
				Structure 3	618	123	0.358	0.435	
Adaptecl3	451650	100	109	Structure 1	896	31	0.065	0.058	142
				Structure 2	420	25	0.089	0.17	
				Structure 3	960	67	0.134	0.126	

design of 65nm technology. From the designers, we know that the blobs (shown as congestion hotspots in Figure 1) were originally ROM blocks, and were late dissolved to ordinary logic circuits to meet the timing closure. Therefore, these GTLs should have dense logic connections according to the designers. Figure 6 indicates that our method successfully finds these logic structures.

In fact, the GTLs captured by our method in Figure 6 match almost exactly with the routing hotspots in upper part of Figure 1, which is from the same design. The characteristics of the solutions are summarized in Table 3. The first column lists the size of each GTL according to the circuit designers. The second column includes the size of the GTL found by our method.

Table 3. GTLs found on the industrial circuit.

Size of GTL in design	Size of GTL found	Cut	GTL-Score
31880	31835	36	0.025
31914	31869	36	0.025
31754	31803	36	0.026
32002	32048	36	0.026
10932	10952	28	0.028

To show the usage of GTLs, all the cells inside the GTLs found through tangled-logic

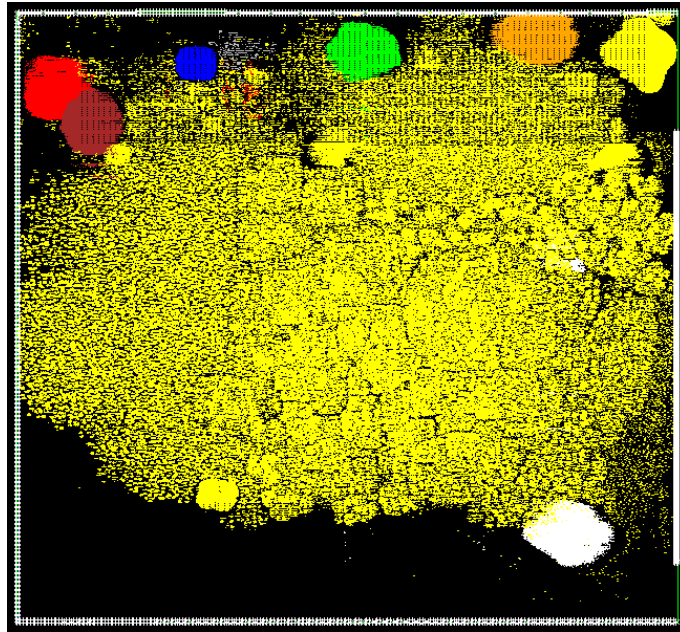


Fig. 6. GTL of the industrial circuit.

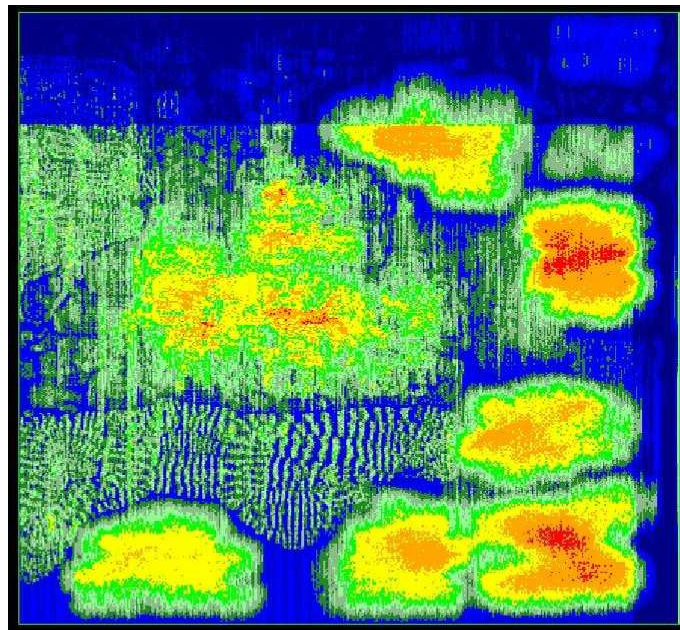


Fig. 7. Routing congestion after cell inflation using GTLs information.

finder algorithm are inflated by four times, and placement was re-performed to spread these cells. Figure 7 shows the routing pictures for this new netlist. Note that since cells are inflated, so the new placement looks different than Figure 1 and Figure 6. It was observed that compared to original placement, the number of nets passing through 100% routing congested tiles are reduced from over 179K to 36K (5X reduction), and the number of nets passing through 90% congested tiles are reduced from 217K to 113K (2X reduction). The average congestion metric ¹ is reduced from 136% to 91%. It is clear that better congestion can be achieved if placement employs cell inflation with GTLs identified with our technique.

¹Measured by taking the worst 20% congested nets and averaging the congestion number of all routing tiles these nets pass through:

CHAPTER VI

CONCLUSIONS

This paper introduces a new problem of finding tangled logic structures from synthesized netlists. These structures can help with floorplanning and routability if special handling is given to cells in these structures. Our new metrics are the first ones to enable the comparison of clusters of different sizes and are normalized so that one can develop standards for tangled logic across a variety of netlists. We demonstrate a possible algorithm for discovering these structures and show how simply inflating the corresponding cells leads to much better routability after placement.

Future work seeks to expand the metrics to handle more specialized structures driven by select lines, and to figure out new ways to use groups of tangled logic to drive better physical design solutions.

REFERENCES

- [1] T. Chen, T. Hsu, Z. Jiang and Y. Chang. "NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs." In *Proceedings of the International Symposium on Physical Design*, 2005, pp. 236-238.
- [2] T. Chan, J. Cong, J. Shinnerl, K. Sze and M. Xie. "mPL6: enhanced multilevel mixed-size placement." In *Proceedings of the 2006 International Symposium on Physical Design*, 2006, pp. 212-214.
- [3] C. J. Alpert, A. B. Kahng, G.-J. Nam, S. Reda and P. Villarrubia. "A semi-persistent clustering technique for VLSI circuit placement." In *Proceedings of the ACM International Symposium on Physical Design*, 2005, pp. 200-207.
- [4] C. J. Alpert and A. B. Kahng. "Recent directions in netlist partitioning: a survey." *Integration: the VLSI Journal*, vol. 19, no. 1-2, pp. 1-81, 1995.
- [5] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. "Spectral K-way ratio-cut partitioning and clustering." In *Proceedings of the ACM/IEEE Design Automation Conference*, 1993, pp. 749-754.
- [6] T.-K. Ng, J. Oldfield, and V. Pitchumani. "Improvements of a mincut partition algorithm." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1987, pp. 470-473.
- [7] L. Hagen and A. B. Kahng. "A new approach to effective circuit clustering." In *Proceedings of the IEEE International Conference on Computer-Aided Design*, 1992, pp. 422-427.
- [8] J. Garbers, H. J. Promel and A. Steger. "Finding clusters in VLSI circuits." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1990, 520-523.
- [9] J. Cong and S. K. Lim. "Edge separability-based circuit clustering with application to multi-level circuit partitioning." *IEEE Transactions on Computer-Aided Design*, vol. 23, no. 3, pp. 346-357, March 2004.
- [10] P. Kudva, A. Sullivan, and W. Dougherty. "Metrics for structural logic synthesis." In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2002, pp. 551-556.
- [11] C. J. Alpert and A. B. Kahng. "A general framework for vertex orderings with applications to circuit clustering." *IEEE Transactions on VLSI Systems*, vol. 4, no. 2, pp. 240-246, June 1996.
- [12] G.-J. Nam, C. J. Alpert, and P. G. Villarrubia. "ISPD 2005/2006 Placement Benchmarks." in *Modern Circuit Placement Best Practices and Results*, pp 3-12, New York: Springer, 2007.

VITA

Tanuj Jindal received his B.Tech. degree in electrical engineering from the Indian Institute of Technology (IIT) Roorkee, India in July 2006. After receiving his Bachelor's degree he worked as a Design Engineer in SOC design group of Wireless Division at Texas Instruments until July 2008. He entered Texas A&M University in August 2008 to pursue his master's degree in computer engineering and received his M.S. in August 2010. His current research is focused on clustering and logic structure identification in VLSI netlists.

Publications:

- T. Jindal, C. J. Alpert, J. Hu, Z. Li, G. Nam, C.W. Winn. “*Detecting Tangled Logic Structures in VLSI Netlists.*” In Proceedings of the ACM/IEEE Design Automation Conference (DAC), 2010, Anaheim, CA. .
- R. Kumar, V. Karkala, R. Garg, T. Jindal, S. Khatri. “*A Radiation Tolerant Phase Locked Loop Design for Digital Electronics.*” In Proceedings of the IEEE International Conference on Computer Design (ICCD), Oct 4-7, 2009, Lake Tahoe, CA.

Tanuj Jindal may be reached at:

tanujjindal@gmail.com

Department of Computer Engineering

c/o Dr. Jiang Hu

Texas A&M University

College Station, TX 77843-3128