

## Generation of Tightly Controlled Circuit Classes for Problems in Physical Design

Debabrata Ghosh   Franc Brglez   Matthias Stallmann

March 2000

2000-TR@CBL-01-Ghosh, Version 1.0

Reprinted, with permission, from  
<http://www.cbl.ncsu.edu/publications/>

Publications at this site are occasionally revised,  
please check for the latest version under the same title.

For more information about CBL, visit  
<http://www.cbl.ncsu.edu/> or write to  
[info@cbl.ncsu.edu](mailto:info@cbl.ncsu.edu)

©2000 authors and CBL, All Rights Reserved

## ABOUT THIS DOCUMENT

*Abstract*— This paper is motivated by the need to generate classes of circuits that are closely related. Ideally, such circuits not only have identical size and distributions of cells and nets but also resemble each other closely in terms of cell clusters and interconnect patterns. Such classes are essential for well-defined performance evaluation of layout algorithms. For example, if the heuristic in the algorithm is being optimized for multiplier structures, then all circuits in the class should have a multiplier-like structure. Such tight control of circuit class generation has not been demonstrated with recent techniques, including the generation of clone and mutant classes.

We introduce an approach to generating circuit classes called *siblings*. Siblings retain all significant characteristics of their parent circuit. We demonstrate that siblings inherit the essential characteristics of the parent only if we control, for each connected component, the diameter of the clusters and the arrangement of the nodes in the underlying spanning tree. We demonstrate the scalability of the proposed approach through a number of experiments. All sibling classes are shown to have results within 5-10% of the parent circuit – under all five physical design tools at our disposal.

**Note.** This document has been published for viewing on the Web in PDF, Postscript and HTML formats. The latter is annotated with active hyperlinks to facilitate quick browsing of this and related documents.

**Citation.** If you choose to cite this report, please add the following entry to your bibliography database:

```
@techreport{
2000-TR@CBL-01-Ghosh,
author = "D. Ghosh and F. Brglez and M. Stallmann",
title = "{Generation of Tightly Controlled Circuit Classes
for Problems in Physical Design}",
institution = "{CBL, CS Dept., NCSU, Box 8206,
Raleigh, NC 27695}",
number = "2000-TR@CBL-01-Ghosh",
month = "March",
year = "2000",
note = "{Also available at
{\tt http://www.cbl.ncsu.edu/~publications/~\#2000-TR@CBL-01-Ghosh}}"}
}
```

**Acknowledgments.** D. Ghosh and F. Brglez have been supported by contracts from the DARPA/ARO (P-3316-EL/DAAH04-94-G-2080 and DAAG55-97-1-0345), and a grant from Semiconductor Research Corporation.

## CONTENTS

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Background and Motivation</b>	<b>3</b>
<b>3</b>	<b>Bigraph Characterization and Equivalence Class Synthesis</b>	<b>4</b>
<b>4</b>	<b>Multi-level Netlist Characterization and Equivalence Class Synthesis</b>	<b>6</b>
<b>5</b>	<b>Additional Experimental Results</b>	<b>8</b>
<b>6</b>	<b>Conclusions</b>	<b>10</b>

## LIST OF FIGURES

1	Illustrating the difficulty in the generation of benchmark equivalence classes: (a) (from top to bottom) drawings of a reference circuit and its clone and mutant, (b) results of experiments using three algorithms on these benchmark classes. The crossing numbers, as reported by dot, are 883 (reference), 610 (clone) and 1503 (mutant). . . . .	3
2	Goal for the generation process in terms of distributions of the isomorphism class and the generated class. . . . .	4
3	Experimental design to compare three isomorphism classes based on three single connected component bigraphs. Each graph has 17 nodes at layer 0, 10 nodes at layer 1, and 50 edges. . . . .	5
4	Experiment showing the proximity of the sibling classes to the two underlying isomorphism classes that have very different crossing number distributions. . . . .	6
5	A simple example illustrating the concept of the CC-graph of a circuit. . . . .	6
6	Pseudo-code for sibling generation process in multi-layer graphs. . . . .	7
7	Comparison of the sibling equivalence class with the underlying isomorphism class: the classes are clearly very similar under all three different tools. This is in contrast to the classes in Figure 1. . . . .	8
8	Summary (min, avg, max) of experimental results with iso, clone, mutant and sibling classes for three ‘difficult’ reference netlists. hmetis (H) and prop (P) reporting mincut and dot reporting crossing number. . . . .	8
9	Summary of experimental results with Rent’s exponent. . . . .	9
10	Comparison of isomorphism, clone and sibling classes under five different tools. . . . .	9

# Generation of Tightly Controlled Circuit Classes for Problems in Physical Design

Debabrata Ghosh

Franc Brglez

Matthias Stallmann

CBL (Collaborative Benchmarking Laboratory)

Department of Computer Science

Box 8206, NC State University,

Raleigh, NC 27695, USA

<http://www.cbl.ncsu.edu/demos>

*Abstract*— This paper is motivated by the need to generate classes of circuits that are closely related. Ideally, such circuits not only have identical size and distributions of cells and nets but also resemble each other closely in terms of cell clusters and interconnect patterns. Such classes are essential for well-defined performance evaluation of layout algorithms. For example, if the heuristic in the algorithm is being optimized for multiplier structures, then all circuits in the class should have a multiplier-like structure. Such tight control of circuit class generation has not been demonstrated with recent techniques, including the generation of clone and mutant classes.

We introduce an approach to generating circuit classes called *siblings*. Siblings retain all significant characteristics of their parent circuit. We demonstrate that siblings inherit the essential characteristics of the parent only if we control, for each connected component, the diameter of the clusters and the arrangement of the nodes in the underlying spanning tree. We demonstrate the scalability of the proposed approach through a number of experiments. All sibling classes are shown to have results within 5-10% of the parent circuit – under all five physical design tools at our disposal.

## 1 INTRODUCTION

There has been a resurgence of research directed to creating larger and more varied benchmark circuits that could test and improve the heuristics in algorithms for physical design [1, 2, 3, 4, 5, 6, 7]. This research complements the traditional approaches to comparing the performance of two or more algorithms which rely on graph equivalence classes, generated as random instances under relatively loose constraints, *e.g.* the random graphs used to evaluate partitioning, coloring, and TSP algorithms in [8, 9, 10], or the generators in Stanford GraphBase [11] used to evaluate the

crossing number minimization algorithms [12], etc.

The exclusive use of random graphs to compare the performance of algorithms has drawbacks: we may fail to detect the difference in performance of two algorithms when one exists; we may bias the design of an algorithm for data sets that will seldom if ever be encountered in applications where a specific cost function minimization is truly important. For example, a placement algorithm optimized for a class of random graphs may not necessarily do as good a placement and wire crossing minimization of relatively sparsely interconnected modules arising in a complex VLSI circuit design.

On the other hand, the problem of generating equivalence classes of graphs that are representative of graphs representing realistic design cases is hard. The solutions proposed in [1, 2, 3, 4, 5, 6, 7] are a major advance towards generating graph structures that represent realistic design cases. However, a more detailed analysis reveals definitive application-specific limitations of current solutions [6]. While new solutions to improve the generation of benchmark classes are being proposed, an unbiased approach to evaluate the relative quality of the proposed benchmark classes is yet to be formalized.

A controlled design of an experiment that measures the performance of a graph-based algorithm requires an equivalence class of graphs that have very similar properties. The fact that a *graph isomorphism class* can also serve for this purpose has been recognized for some time [13]. Changing the starting point for the problem instance can and *does* induce unpredictable and *significant* variability of reported cost functions when the experiments are repeated.

The graph classes considered in this paper are abstractions of design- and domain-specific netlists, *i.e.* directed hypergraphs. Two netlists, one representing a multiplier-like device, the other a specific controller device, may have the same number of inputs/outputs, the same number, the same size, and the same distributions of cell nodes, and nets (hyperedges), and yet they will differ significantly in the layout area and total wire length and wire crossing after embedding onto a plane and routing all nets under technology-specific design rules. We introduce the notion of *reference netlist*, such as a multiplier or a controller, to induce a distinctive *isomorphism class*.

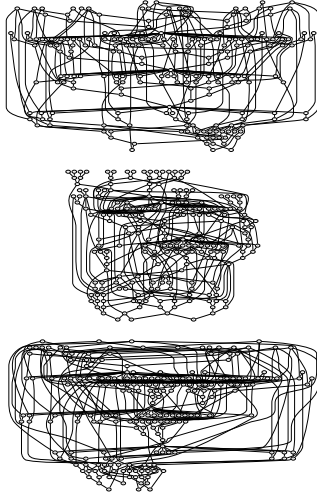
This paper addresses two issues. First, we propose that

---

\* D. Ghosh and F. Brglez have been supported by contracts from the DARPA/ARO (P-3316-EL/DAAH04-94-G-2080 and DAAG55-97-1-0345), and a grant from Semiconductor Research Corporation.

“Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of CBL. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.”

(a) Graph drawings by dot [14]



(b) Experiments with 3 algorithms and 3 equivalence classes of netlists.

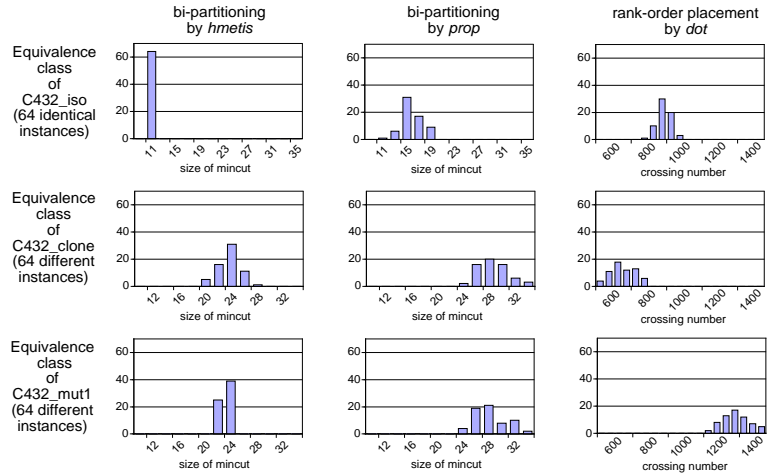


Fig. 1. Illustrating the difficulty in the generation of benchmark equivalence classes: (a) (from top to bottom) drawings of a reference circuit and its clone and mutant, (b) results of experiments using three algorithms on these benchmark classes. The crossing numbers, as reported by `dot`, are 883 (reference), 610 (clone) and 1503 (mutant).

any new class, generated from the characterization of the underlying reference netlist, be evaluated and compared to the respective isomorphism class. This comparison should engage as many algorithms as possible; ideally *all* the cost functions distributions reported by the algorithms should have means that are comparable to the mean of the isomorphism class.

In addition, we propose a new approach to generating a new class: the *siblings* class of circuits based on a representative parent circuit. We demonstrate that siblings inherit the essential characteristics of the parent only if we control, for each connected component, the diameter of the clusters and the arrangement of the nodes in the underlying spanning tree. We demonstrate the scalability of the proposed approach through a number of experiments. All sibling classes are shown to have results within 5-10% of the parent circuit – under *all* five physical design tools at our disposal.

The paper is organized into five major sections. Section 2 provides background and motivation, Section 3 introduces bigraph characterization and equivalence class synthesis, Section 4 formalizes the equivalence class synthesis of multi-level netlists, Section 5 summarizes a large number of experiments.

## 2 BACKGROUND AND MOTIVATION

We use a simple experiment in Figure 1 to illustrate the issues involved in generating equivalence classes that can be used for performance evaluation of algorithms. The three algorithms are: `hmetis` [15] and `prop` [16] for balanced netlist bi-partitioning, and `dot` for rank-order directed graph placement [14]. The three equivalence classes are the isomorphism class as introduced in [13], the class of clones [4], and the class of mutants [5]. All these classes

correspond to the benchmark circuit C432.

Briefly, we create the netlist isomorphism class  $\mathcal{N}_{iso}$  as follows: (1) take a reference netlist represented as a hypergraph  $G_r(V, E)$ , (2) apply, uniformly to all nodes in  $G_r$ , a *random re-order, and random re-label* procedure  $rr(V)$ :

$$\mathcal{N}_{iso} = \{N_j \in G_r(rr(V), E)\} \quad (1)$$

Relative to all other instances in  $\mathcal{N}_{iso}$ , each instance  $N_j$  has the following properties:

- P1: the order of nodes in  $N_j$  is uniformly random;
- P2: the labels of nodes in  $N_j$  are uniformly random.

The class of clones [4] relies on a specific netlist characterization which must be satisfied to form an equivalence class of related ‘clone circuits’. The class of mutants [5] differs from the class of clones by way of the netlist characterization and the constraints that must be satisfied when generating a ‘mutant circuit’.

Nominally, each instance of a netlist from *any* equivalence class is related to a *reference netlist* that may typically represent an application-specific design rather than a netlist generated randomly. It is difficult for the generation algorithm to replicate the characteristics of the netlist. Drawings (by `dot`) in Figure 1(a) of the instances from the isomorphism, clone and mutant classes show that the clone and the mutant both are different from the reference netlist. This is further confirmed by the crossing numbers of the graphs which are 883, 610 and 1503 for the reference, clone and mutant respectively.

Results of the experiments with `hmetis`, `prop` and `dot` are shown in Figure 1(b). Observations about each equivalence class follow the figure from the top down:

*Isomorphism class:* An ideal algorithm should produce a distribution with variance 0 with the isomorphism class. Although `hmetis` returns a mincut of 11 for all 64 instances

in this class, the distributions, with relatively large variance, observed for `prop` and `dot` are what may be observed typically, especially when the size of the circuit increases.

*Class of clones:* The centers of distributions induced by this class have clearly shifted significantly w.r.t. the distributions of the isomorphism class. One can argue that the netlist in this class are far from similar to the isomorphism class of the reference netlist.

*Class of mutants:* Again, one can argue that the netlist in this class are far from similar to the isomorphism class of the reference netlist.

The rhetorical question at this point is:

*Which of these classes, if any, is best suited to evaluate the comparative performance of each algorithm?*

We argue that the isomorphism class is clearly one that exposes a number of problem areas with a design of a heuristic that is being evaluated. Neither the clone class nor the mutant class are representative (in the cases shown) of the underlying isomorphism class. We may learn more about the limitations of the heuristic by looking for another reference circuit and create an isomorphism class with the center of distribution shifted from the first class. However, finding an equivalence class of *different netlists* but a distribution that is similar to the underlying isomorphism class of the reference circuit has a number of merits such as (1) increased diversity of subjects, but within well-defined bounds, and (2) increased number of tests by creating additional isomorphism classes of each distinct subject. The question that remains to be answered is how to judge the quality of such a class.

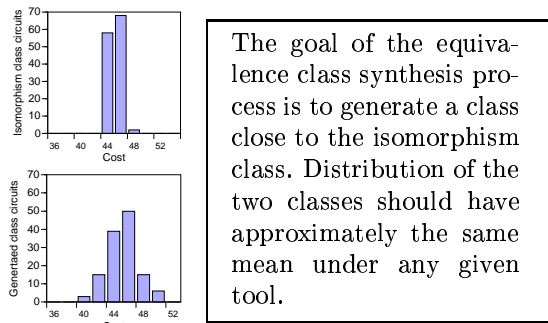


Fig. 2. Goal for the generation process in terms of distributions of the isomorphism class and the generated class.

We believe that evaluation of an equivalence class generated from reference netlist should be based on the proximity of the generated class to the underlying isomorphism class. This is illustrated in Figure 2. The two classes should behave in approximately the same way under any given tool. This implies that *under all the tools*, the mean cost for the two distributions should be within (a predetermined)  $\pm\epsilon_0\%$ .

In this paper we pursue this goal in steps. First, we formalize the process for two-layer graphs (bigraphs) since the problem is difficult for bigraphs alone. Later, we extend

the method to multi-layer graphs representing electrical netlists.

### 3 BIGRAPH CHARACTERIZATION AND EQUIVALENCE CLASS SYNTHESIS

The failure of the clone and mutant equivalence classes to emulate the characteristics of the isomorphism class in Figure 1 leads us to believe that the characterization of the reference netlist is not complete in these two approaches. The goal of this work is to perform a more comprehensive characterization of the circuit and propose a method for generating the corresponding equivalence class. The problem is far from being easy for a bigraph itself as has been seen in a related work reported elsewhere. More importantly, the capabilities developed for a bigraph can be directly used for multi-level electrical netlists since we model the latter as a concatenation of bigraphs. Hence, a successful sibling generation process for bigraphs is a prerequisite for success in sibling generation for multi-level netlists.

**Bigraph Characterization.** Depending on the characterization process, we define two types of signatures, basic and full, for a bigraph.

*Basic Signature:* Complexity of a bigraph class may be characterized by three different parameters. In place of the usual  $n_0$  = number of layer-0 nodes,  $n_1$  = number of layer-1 nodes, and  $m$  = number of edges, we look at a *basic signature*  $(a, b, m)$ , where  $a = m/(n_0 + n_1 - 1)$  (total edges/spanning-tree edges),  $b = n_0/n_1$  (balance factor), and  $m^1$ .

This is far from sufficient, however, if we want meaningful experimental results that are applicable to subjects in a given class. Consider, for example, the bigraphs shown in Figure 3. All belong to the class with basic signature  $(\frac{25}{13}, 1.7, 50)$ , i.e.  $n_0 = 17$ ,  $n_1 = 10$ , and  $m = 50$ , but each is generated differently. The instance of the graph in (a) is a single connected component generated randomly via [11]. The instances of the graph in (b) and (c) are single connected components designed to have low and high crossing numbers, respectively. We use each graph to generate a 64-subject isomorphism class. Two heuristics `dot` [14], and `TR17` [17], are applied to each class. Evidently, the three classes possess very different characteristics in terms of wire crossing although they share the same basic signature.

This leaves us with a fundamental question: how can we distinguish between graphs that have the same basic signatures but different crossing numbers and can the distinction be used to generate equivalence classes with similar characteristics? We propose further characterization of the graph in the form of its *full signature*.

*Full Signature of a bigraph:* We conjecture that the underlying structure of a graph (at least w.r.t. crossing number) can be characterized by its spanning trees. The exact nature of the relationship is still being investigated but it appears to depend on the first two parameters of our basic signature. We have seen, through extensive experiments, that

<sup>1</sup>The parameters  $(a, b, m)$  convey more information (those related to spanning trees) than  $(n_0, n_1, m)$ , and,  $n_0, n_1, m$  can be derived from the triplet  $(a, b, m)$

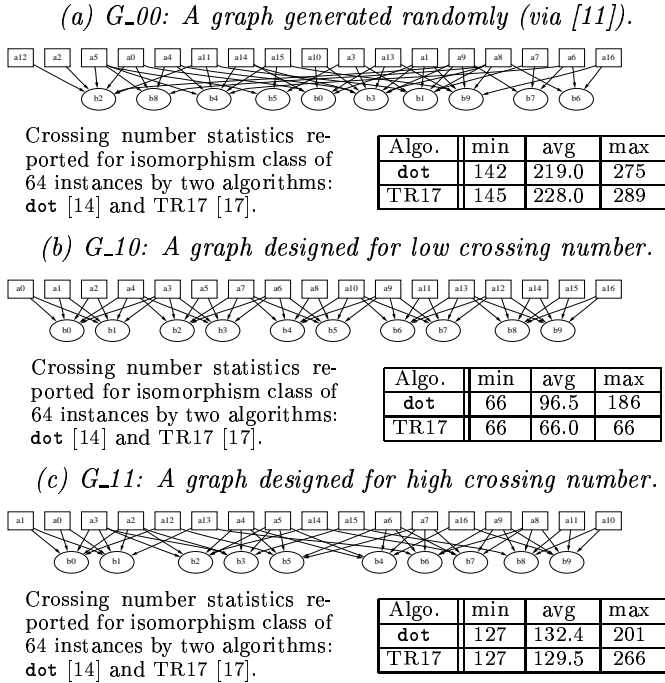


Fig. 3. Experimental design to compare three isomorphism classes based on three single connected component bi-graphs. Each graph has 17 nodes at layer 0, 10 nodes at layer 1, and 50 edges.

there is a relationship between the average crossing number of an isomorphism class and that of the *characteristic spanning tree class* (consisting of a number of randomly selected spanning trees of the graph). Reference graphs with smaller crossing numbers lead to characteristic spanning tree classes with smaller average crossing numbers.

The full signature is constructed in the following way. We select 64 spanning trees of the graph at random and optimize each of them using TR17<sup>2</sup>. Then, we choose the spanning tree that has a crossing number closest to the average of the spanning tree class. The order of cell and net nodes in this spanning tree, as determined by TR17, is now added to the signature. The full signature, thus, is defined as:

$$\{(a, b, m), \{\kappa, \nu\}\}$$

where  $\kappa$  and  $\nu$  are the order of cell nodes and net nodes respectively for the minimum crossing spanning tree of the graph. The full signature captures most attributes of the graph, and hence, the equivalence classes generated from it will be, hopefully, the most representative of the reference graph.

**Sibling Class Synthesis.** We are now ready to present a method of generating a class of siblings from a reference graph  $G$ .

<sup>2</sup>Theoretically we could use an exact algorithm since the crossing number problem can be solved in polynomial time for trees, albeit via a complicated algorithm [18]. However, for all practical purposes, TR17 does a good job for trees.

- First, we re-order the canonical form based on the spanning tree and the cell and net order returned by TR17. The resulting ordering becomes the basis for a clustered approach to adding extra edges randomly.

- Then, we reconnect all the edges subject to the following constraints

- (1) fan-in constraint of the cell nodes,
- (2) fanout bounds on the net nodes,
- (3) clustering diameter.

#### Clustering: Additional Control for Graph Synthesis.

The reason why random graphs cannot be a useful alternative to the isomorphism class generated from the reference graph is that random graphs fail to follow the specific interconnection pattern that may be present in the reference graph. This was first observed in [4]. In particular, in the beginning of generating a random bipartite graph there is an equal probability of each node in vertex set 1 to be connected to each node in vertex set 2, irrespective of the size of the vertex sets. This is not true for most graphs seen in real life and especially for graphs representing VLSI circuits, where the probability of connection gets infinitesimally small for distant nodes. If we do not maintain this property, the equivalence class generated will be far away from the isomorphism class. We propose a clustering method to take care of the gradually diminishing probability of connection for distant nodes. The method works as follow:

- We define a *clustering diameter*  $D$  which is the maximum allowable span for connection<sup>3</sup>.
- Each cell (net) node  $n$  is assigned an index number,  $n_i$ , which is its position in the spanning tree.
- For each cell (net) node  $n$ , the natural ally  $a$  is defined as a net (cell) node that has the index number closest to  $n_i \times r$  where  $r$  is the ratio of number of cell and net nodes.
- For any connection to be done, we define a *feasible set*  $F_s$  that contains all the possible alternatives.
- We define a cluster set  $F_c$  as  $(D + 1)$  consecutive nodes with  $a$  approximately at the center. So, the cluster consists of  $a$  at its the center and  $D/2$  nodes before and  $D/2$  nodes after  $a$ .
- If  $F_c \cap F_s \neq \emptyset$ ,  $F_s$  is assigned the value  $F_c \cap F_s$ .
- Otherwise, we keep increasing  $D$  until  $F_c \cap F_s$  ceases to be  $\emptyset$ . Then, assign  $F_s = F_c \cap F_s$ .
- Note that if a feasible set is available, clustering method always returns a set of valid possible connections. It attempts to confine the connections within the cluster, but adjusts the cluster diameter dynamically when other constraints preclude connections within the cluster. Hence, it does not introduce any extra constraints that are binding.
- The final connection will be chosen at random from this (pruned) feasible set.

<sup>3</sup>The span is defined as the maximum difference between the cell (net) nodes that the net (cell) node is adjacent to. The distance is measured in terms of the position of the nodes in a linear order. The linear order is the order of net and cell nodes in the optimized spanning tree in this case.

*Need for the Spanning Tree Order:* The spanning tree order is needed for two reasons:

- The clustering method is based on some pre-defined ordering of cell and net nodes. We use the spanning tree order since one can determine it accurately. Absence of any well-optimized order may render the clustering method ineffective as has been seen elsewhere.
- The spanning tree and its node orders are the only parameters responsible for differentiating between two graph classes with the same basic signature but very different crossing number characteristics.

In essence, it is the combination of the clustering method and the spanning tree order that helps us generate excellent sibling equivalence classes such as those in Figure 4.

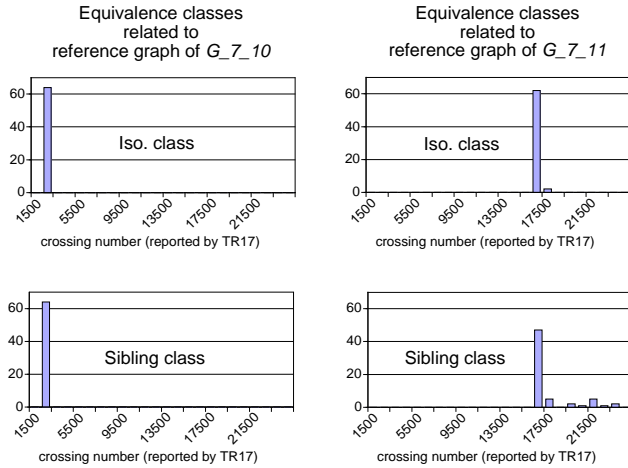


Fig. 4. Experiment showing the proximity of the sibling classes to the two underlying isomorphism classes that have very different crossing number distributions.

**Snapshot of experiments.** Figure 4 shows that the new sibling class does a remarkably good job of replicating the crossing number distributions for the isomorphism classes of the two reference graphs  $G_{7_10}$  and  $G_{7_11}$  which have very different crossing numbers. The equivalence classes generated from  $G_{7_10}$  and  $G_{7_11}$  are different under TR17 and are close to the respective equivalence classes. No other method induces different equivalence classes for  $G_{7_10}$  and  $G_{7_11}$ .

For the reference graph  $G_{7_10}$  we were tempted to suspect that the subjects in the sibling class were all isomorphic to the reference graph. Not so. The 64 random siblings used to obtain the singular distribution in the lower left histogram exhibited 53 distinct degree sequences! Hence, 53 siblings out of 64 are decidedly *different from the reference graph* and *different from each other*. Question of isomorphism is undecided for the rest of the 11 graphs.

#### 4 MULTI-LEVEL NETLIST CHARACTERIZATION AND EQUIVALENCE CLASS SYNTHESIS

In this section we extend the sibling synthesis procedure to graphs representing multi-level electrical netlists. As was mentioned before, a successful synthesis method for

bigraphs is only a *necessary* and *not sufficient* condition for success in multi-level netlist equivalence class synthesis. Multi-level netlists require special characterization which we will discuss in this section. For simplicity, we will restrict our discussion to combinational circuits only. The process itself is intricate and complex. Only essential procedures can be outlined in the limited space available. For more details see [19].

**Canonical form.** An electrical netlist, which is a hyper-

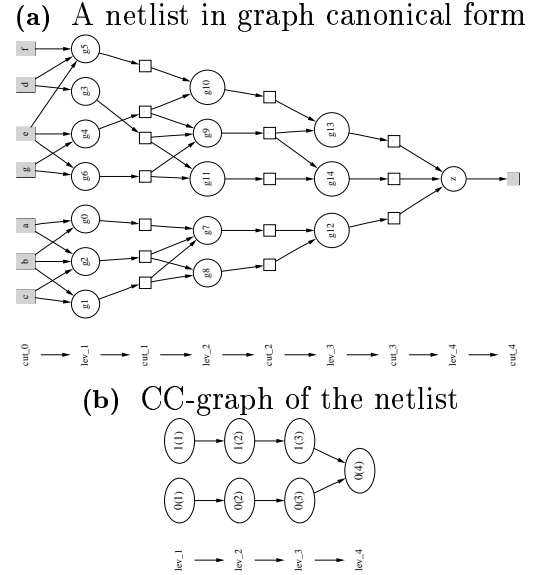


Fig. 5. A simple example illustrating the concept of the CC-graph of a circuit.

graph, is first converted into a  $2k$ -partite directed graph through a series of transformations. This is called a *canonical form* of the circuit. Here,  $k$  is the maximum logic level in the circuit. Figure 5(a) shows the canonical form of a circuit as drawn by dot. Note the bipartite nature of the graph induced by cell nodes (circles) and net nodes (rectangles).

The  $2k$ -partite graph canonical form is produced through the following steps.

*First Levelization:* This procedure is well known.

- All primary inputs (PIs) are assigned a  $level = 0$ .
- All cells are assigned a  $level > 0$  using topological sort.
- Each net inherits the name and level of the driving cell.

*Second levelization:* The initial levelized form is further modified as follows:

- Each PI net is assigned a  $level = l - 1$  where  $l$  is the minimum level of cells that it is driving.
- Each net is assigned a netspan  $\lambda = p_{max} - p_{min}$  where  $p_{max}$  and  $p_{min}$  are the maximum and minimum level of cells it is adjacent to.
- Each net of netspan  $\lambda > 1$ , is broken into segments by introducing  $\lambda - 1$  single-input, single-output feedthroughs. Feedthrough at level  $i$  is named as  $\langle net\_name \rangle . i$ .

- Finally, each net is replaced by a net node with fan-in of 1, but variable fanout.

*Finding the local connected components at each level:*

We capture the local clustering by finding the *connected components (CCs) of the underlying undirected graph* at each level. This is done by a simple DFS search. CCs at level  $i$  consists of cell nodes at level  $i$  and net nodes at level  $i - 1$ . Note that these CCs are local to each slice and not a connected component of the entire graph.

*Forming the CC-graph:* The local CC's induce a  $k$ -partite graph called the *CC-graph* with an edge between a CC  $C_{i-1}$  in level  $i - 1$  and  $C_i$  in level  $i$  whenever  $C_{i-1}$  has a cell connected to a net of  $C_i$ . Figure 5(b) shows the CC-graph of the circuit. In this figure, the CC nodes are represented in the form  $\langle \text{CC\_index}(\text{level}) \rangle$ .

*Capturing the hierarchy through the CC-graph:* The CC-graph is an attempt to capture the hierarchy of the circuit. The canonical form depicts the flat netlist and fails to capture any hierarchy or macro-structure in the circuit. We attempt to model the hierarchy in the circuit by modeling the local clustering present in each level in the form of the CCs. In the presence of a hierarchical structure in the graph most of the wire crossing will be present inside the CCs and the CC-graph will show relatively few wire crossing compared to the original graph. We maintain the CC-graph for all the siblings that we generate from the canonical form of the reference graph. Hence, all the circuits in the equivalence class are likely to maintain the hierarchical structure that was present in the reference graph. This is in contrast to the mutants [5] or the clones [4] where there was no mechanism to preserve the hierarchy of the circuit as noted in [6].

**Characteristic Signature for Multi-level Graphs.**

The signature of the multi-level graph is essentially a 2-dimensional version of the signature of bigraphs. However, one needs to include:

1. the concept of CC-graph
2. three types of net nodes that may be present in a multi-level graph:
  - Type  $I$ : primary inputs of the circuit
  - Type  $C$ : driven by logic cell nodes that are not feedthroughs
  - Type  $A$ : driven by feedthrough cells

Consequently, the signature for the multi-level graph will be  $\{A, B, M, \Lambda_A, N_I, N_O\}$ , where  $A, B, M$ , list the  $a, b, m$ , values at all level,  $\Lambda_A$  lists the number of feedthrough nodes at all levels, and  $N_I, N_O$  list the primary inputs and outputs at all levels.

*Full signature:* The full signature needs the spanning tree ordering information and the ordering of the CC-graph. Providing spanning tree ordering information for a multi-level graph is complicated. This is because the spanning trees are for the CCs at each level and wire crossing in CC  $j$  at level  $i$  is dependent on the net node ordering in CCs at level  $i - 1$ . Consequently, a spanning tree order at level  $i$  without any history of the arrangements upto level  $i - 1$  does not help much in obtaining a good order for the entire multi-level graph. For this reason we provide the spanning

tree orders of the *first level only* in the signature.

Finally, the full signature for the multi-level graph is given by:

$$\text{full\_signature} = \text{basic\_signature} \cup \{\kappa, \nu\} \cup \Psi$$

where  $\{\kappa, \nu\}$  is the order of the net nodes and cell nodes respectively *at level 1* and  $\Psi$  is the ordering of the CC-graph as reported by dot.

**Sibling Synthesis Process.** The algorithm for the sibling synthesis for multi-layer graphs is basically an extension of the connection procedure for bigraphs, with modifications to take care of

- (1) the increased number of constraints due to the presence of different types of nodes,
- (2) concept of ordering in CC-graph,
- (3) dependence of node ordering in CC at level  $i$  on the node orders in CCs at level  $i - 1$ .

```
function gen_sib() {
    read in the 2k-partite canonical form;

    read in the dot-optimized order of CC-graph and order
        the local CCs at each level accordingly;

    perturb the canonical form;

    for each level i {
        for each CC j {
            if(i==1) {
                Read in the order of net nodes and cell nodes;
                connect_bigraph();
            }
            else {
                Form a spanning tree at random;
                order_net_nodes();
                Use (single pass) barycenter to order cells;
                connect_multi_layer();
                Use (single pass) barycenter to order cells again;
            }
        }
    }
}

function order_net_nodes(){
    for each net n {
        c = cell node that drives n;
        CC_c = CC that c belongs to;
        y1 = position of CC_c in CC-graph
        y2 = position of c within CC_c
        y = y1*10^8 + y2 /*assume, y2<10^8*/
    }
    sort nets based on their y-value using barycenter
}

function connect_multi_layer(){
    Each node of type C, give connection from a net of type
    C at level i-1 (using cluster diameter of 3);
    connect_bigraph();
}

function connect_bigraph() {
    /* same as the connection procedure for bigraphs */
    /* It uses a cluster diameter of 3. */
}
```

Fig. 6. Pseudo-code for sibling generation process in multi-layer graphs.

The process is complicated and we present the essence of

the algorithm in the pseudo-code in Figure 6. The following points are to be noted about the algorithm:

- The process is initiated by deleting all the wires in the canonical form.
- Re-connecting these edges, under various constraints, constructs the sibling circuit.
- The order of both the net nodes and cell nodes is important for the clustering method to succeed. The modifications in the connection procedure for multi-layer graphs is aimed at *producing a good order*.
- Ordering of net nodes in CC  $j$  at level  $i$  depends on ordering of the cell nodes that drive these net nodes. This is done by the function `order_net_nodes` in Figure 6. Note that the driving cell nodes may belong to different CCs in level  $i - 1$ . So, the net nodes in CC  $j$  at level  $i$  are sorted first based on the CC numbers of the driving cell nodes, and then, based on position of the cells within the respective CCs at level  $i - 1$ . (This is possible due to a one-to-one correspondence between the net nodes and the cell nodes that drive them).
- Level 1 is a special case. Since no ordering is imposed by a previous level, TR17 is used on the spanning tree of each CC to get an ordering of minimal crossing number before adding extra edges via clustering.
- At each level  $i > 1$ , the order of the net nodes is determined by their driving cell nodes, and hence, cannot be changed by any crossing minimization algorithm. So, the spanning tree is optimized by a single pass of barycenter method which is probably close to optimal for two-layer trees with one layer fixed.
- The function `connect_multi_layer` takes care of the additional complexity in the multi-layer graph.
- After all the connections are done, we apply one more pass of barycenter method to minimize any crossing that can be avoided.

**Snapshot of experiments.** As a first test, we complete the experiments introduced in Figure 1 earlier. Again, we apply the three algorithms to all 64 netlist instances of the new sibling class: a partitioner `hmetis` [15], a partitioner `prop` [16], and a placer `dot` [14]. The resulting distributions for the new class *c432\_sib* are shown in Figure 7. Clearly, the distributions are very similar to the distributions for the underlying isomorphism classes of the reference netlists. It has also been verified, through the degree sequence test, that all the siblings are *different from the reference circuit* and they are also *different from each other*.

## 5 ADDITIONAL EXPERIMENTAL RESULTS

In this section we present results for additional experiments with the multi-layer graphs. First, we report results for sibling classes of some reference graphs which were ‘difficult’ for both clone and mutant classes in that the mean was significantly higher than that of the respective isomorphism class. We show that the sibling classes are ‘close’ to the isomorphism classes. Then we test how scalable the sibling generation process is, namely, how the sibling classes of large circuits fare w.r.t. to the underlying isomorphism class. We also include results with the clone classes in our

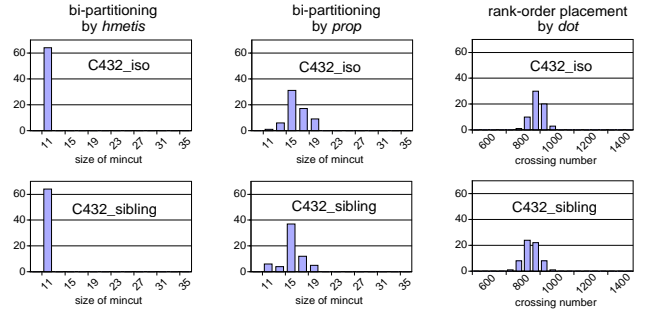


Fig. 7. Comparison of the sibling equivalence class with the underlying isomorphism class: the classes are clearly very similar under all three different tools. This is in contrast to the classes in Figure 1.

comparisons.

**Experiments with ‘Difficult’ Reference Graphs.** The reference netlists C432, C880 and C6288 were chosen since none of the available benchmark generation tools could generate classes sufficiently similar to the respective isomorphism class. All these netlists are from the ISCAS’85 data set [20]. The netlist of C6288 is a model of a 16x16 bit array multiplier. As the results in Figure 8 demonstrate, this reference circuit is a particular challenge for the clone equivalence class. Both the average mincut and the average crossing number of the clone netlists are several orders of magnitude larger than the mincut and the crossing number of the reference netlist.

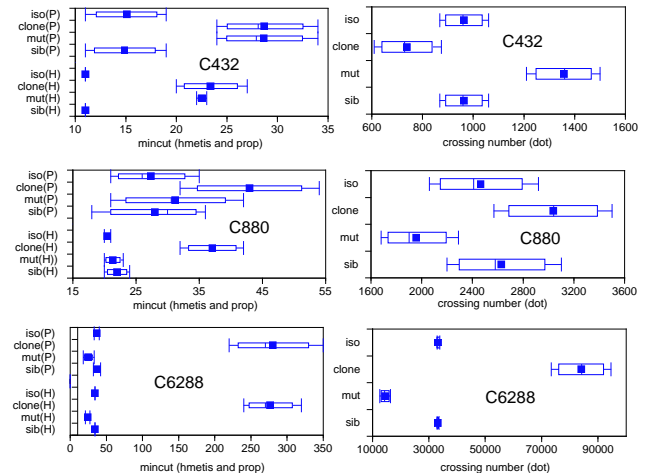


Fig. 8. Summary (min, avg, max) of experimental results with iso, clone, mutant and sibling classes for three ‘difficult’ reference netlists. `hmetis` (H) and `prop` (P) reporting mincut and `dot` reporting crossing number.

On the other hand, Figure 8 shows that all three sibling classes, *C432\_sib*, *C880\_sib* and *C6288\_sib*, have averages as well as min/max values that are very comparable to the ones shown for the respective isomorphism classes.

**Experiments with Rent’s exponent.** Rent’s exponent has been traditionally used as a measure of interconnection

complexity, and, a number of synthetic benchmark generation methods [1, 6, 7] have used this parameter for characterization. Hence, we wanted to observe how the sibling classes behave in terms of the Rent's exponent. In Figure 9, we plot the average, minimum and maximum Rent's exponent for each of the isomorphism, clone, mutant and sibling classes. The Rent's exponent has been determined using `hmetis` [15], with  $k$ -way partitioning ( $k = 16, 32, 64, 128$ ). It may be observed that although the sibling classes have not been generated with any specific goal of maintaining the Rent's parameter, *all* the sibling classes maintain the Rent's parameter of the corresponding isomorphism classes. The mutant classes are close to the isomorphism classes for C880 and C6288, but not so for C432. The clone classes too have problems. The clone class for C6288 is especially bad where the distribution of Rent's exponent is way beyond the range of that for the isomorphism class.

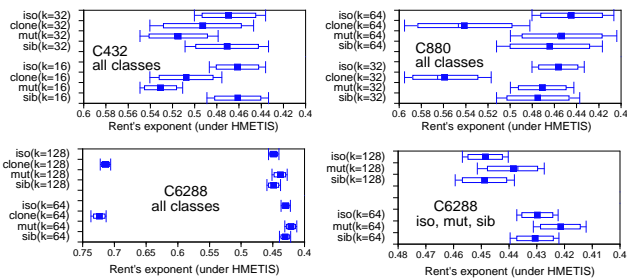


Fig. 9. Summary of experimental results with Rent's exponent.

**Detailed Comparison.** To demonstrate that the observed improvements are not due to chance, we designed similar experiments with additional equivalence classes. These experiments validate the quality of the sibling classes generated and compare them with the respective clone classes (we compare with the clone classes only in Figure 10 since this is the most extensive benchmark generation process that is available in the public domain). These experiments show that the sibling classes satisfy the goals we had set in Figure 2 – all the *sibling classes* are within  $\pm\epsilon_0\%$  of the isomorphism classes *under all the tools used*.

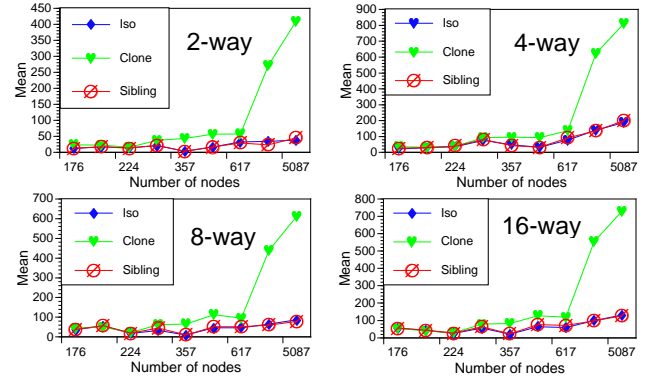
We have selected 9 reference circuits (C432, C499, 9symml, C880, i5, C1355, C1908, C6288, des) from [20] with 176 nodes in the smallest one (C432) to 5087 nodes in the largest (des).

We use the following five tools in these experiments:

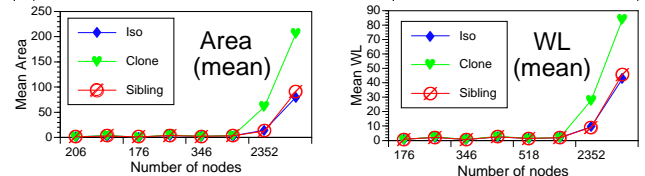
1. `hmetis`[15] for 2, 4, 8, 16-way partitioning
2. `dot`[14] for wire crossing minimization
3. `prop`[16] for bi-partitioning
4. `oasis`[21] for standard cell placement and routing
5. `vpr`[22] for FPGA place and route

Results are shown in Figure 10. Here, we plot the mean of the cost parameters (as reported by each of the five tools) for all three classes – isomorphism, clone and sibling equivalence classes. Evidently, the sibling classes are the closest to the isomorphism class. The clone classes are close too for small circuits, however, as the circuit size increases the clones tend to deviate widely from the isomorphism

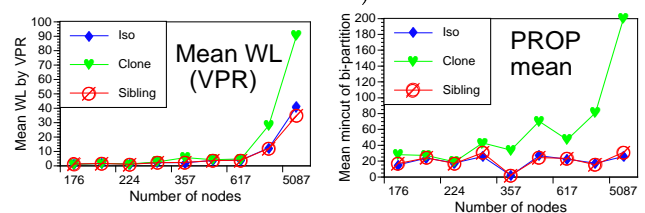
(a) mean of `hmetis` distributions (2-way, 4-way, 8-way 16-way)



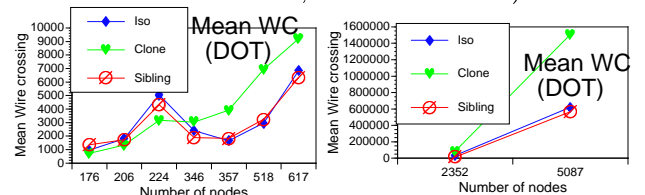
(b) mean of `oasis` distributions (area and wire length)



(c) mean of `vpr` and `prop` distributions (wire length and mincut)



(d) mean wire crossing of `dot` distributions (right plot for C6288 and des, left for all others)



(e) Average unsigned difference (over all 9 classes) w.r.t. the isomorphism class – for clone and sibling classes.

Tool	Diff. in mean <sup>1</sup> for sibling	Diff. in mean <sup>2</sup> for clone	Diff. in s.d. <sup>3</sup> for sibling	Diff. in s.d. <sup>4</sup> for clone
<code>hmetis</code> [15]	5.62%	230.7%	— <sup>5</sup>	— <sup>5</sup>
<code>dot</code> [14]	8.97%	79.66%	94%	330%
<code>prop</code> [16]	9.52%	236%	21%	47%
<code>oasis</code> [21]	5.26%	58.9%	16.6%	85%
<code>vpr</code> [22]	7.47%	53.3%	32.8%	105%

<sup>1</sup> given by  $(1/9) \sum_{i=1}^9 |\mu_{clone,i} - \mu_{iso,i}|$

<sup>2</sup> given by  $(1/9) \sum_{i=1}^9 |\mu_{sib,i} - \mu_{iso,i}|$

<sup>3</sup> given by  $(1/9) \sum_{i=1}^9 |\sigma_{clone,i} - \sigma_{iso,i}|$

<sup>4</sup> given by  $(1/9) \sum_{i=1}^9 |\sigma_{sib,i} - \sigma_{iso,i}|$

<sup>5</sup> undefined since std. dev.=0 for iso. class

Fig. 10. Comparison of isomorphism, clone and sibling classes under five different tools.

classes. This phenomenon has been observed by others too [6]. These figures show that the sibling generation process is scalable, at least for the reference graphs used here.

**Comparison of  $\pm\epsilon_0$  similarity of clone and sibling classes with the isomorphism class.** Finally, we summarize the percentage deviation of the clone/sibling class from the isomorphism class under five different tools. Here, for each tool and for each reference graph, we take the *absolute value of the difference* of the mean and std. deviation with the mean and std. deviation of the respective isomorphism class. We present the average of this absolute value over all the 9 reference graph classes in Figure 10(e) for each of the five tools. Clearly, the sibling classes show an acceptably low difference (5–10%) with the isomorphism classes *for all the classes* and *for all the tools* used in this experiment. The difference of the clone classes from the isomorphism classes is prohibitively high: 50% – 200%.

## 6 CONCLUSIONS

A controlled design of an experiment that measures the performance of a graph-based algorithm requires an equivalence class of graphs that have properties that are similar to well-defined reference graphs, *e.g.* a multiplier or a controller. Experiments have shown that the generation of graph instances without sufficient context to the physical design problem itself does not provide the control which is necessary to create a class similar to its reference. This clearly rules out the generation of graphs by traditional random methods. The recent clone and the mutant classes represent a significant improvement over the traditional random methods of class generation. However, either method can still generate classes that may have no similarity with the reference circuit isomorphism class. Experiments show that a number of clone and mutant classes exceeds the target  $\pm\epsilon_0\%$  similarity test.

The sibling class, introduced in this paper, is based on graph characterization that captures not only the underlying structure of the graph spanning tree, but also the optimized order of its linear arrangement. We created sibling classes from the reference graphs for which the earlier methods failed the  $\pm\epsilon_0\%$  similarity test. The experiments with the new sibling classes demonstrate that *all of them* meet the 5–10% similarity test for *all five* physical design tools available.

## REFERENCES

- [1] J. Darnauer and W. Dai. A Method for Generating Random Circuits and its Application to Routability Measurement. In *4th ACM/SIGDA Int'l Symp. on FPGAs, FPGA96*, pages 66–72, February 1996.
- [2] Michael Hutton, J.P. Grossman, J. Rose and D. Corneil. Characterization and Parametrized Random Generation of Digital Circuits. In *Proceedings of the 33rd ACM IEEE Design Automation Conference*, June 1996.
- [3] Michael Hutton, J.P. Grossman, J. Rose and D. Corneil. Generation of Synthetic Sequential Benchmark Circuits. In *ACM Symposium on FPGAs*, pages 149–155, February 1997. Available from <http://www.eecg.toronto.edu/~pubs/pubs.html>.
- [4] Michael Hutton, J.P. Grossman, J. Rose and D. Corneil. Characterization and Parameterized Random Generation of Combinational Benchmark Circuits. *IEEE Trans. Computer-Aided Design*, 1999. To appear. Postscript available from <http://www.eecg.toronto.edu/~jayar/pubs/pubs.html>. Software available from <http://www.eecg.toronto.edu/~mdhutton/gen/Terms.html>.
- [5] Debabrata Ghosh, Nevin Kapur, Justin E. Harlow III, and Franc Brglez. Synthesis of Wiring Signature-Invariant Equivalence Class Circuit Mutants and Applications to Benchmarking. In *Proceedings, Design Automation and Test in Europe*, pages 656–663, Feb 1998. Also available at <http://www.cbl.ncsu.edu/publications/#1998-DATE-Ghosh>.
- [6] J. Pistorius, E. Legai and M. Minoux. Generation of Very Large Circuits to Benchmark the Partitioning of FPGAs. In *Proc. Intl. Symp. on Physical Design*, 1999.
- [7] D. Stroobandt, P. Verplaeste and J. V. Campenhout. Towards Synthetic Benchmark Circuits for Evaluating Timing-Driven CAD Tools. In *Proc. Intl. Symp. on Physical Design*, 1999.
- [8] D. S. Johnson, C. R. Aragon, L. A. McGeogh, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part I (graph partitioning). *Operations Research*, 37(6):865 – 892, 1989.
- [9] D. S. Johnson. Local optimization and the traveling salesman problem. In *Proc. 17th International Colloquium on Automata, Languages, and Programming*, pages 446 – 461, 1990.
- [10] D. S. Johnson, C. R. Aragon, L. A. McGeogh, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part II (graph coloring and number partitioning). *Operations Research*, 39(3):378 – 406, 1991.
- [11] Donald E. Knuth. *The Stanford Graphbase*. Addison Wesley, 1993.
- [12] M. Jünger and P. Mutzel. 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. *Journal of Graph Algorithms and Applications (JGAA)*, 1(1):1–25, 1997.
- [13] M. R. Hartoog. Analysis of Placement Procedures for VLSI Standard Cell Layout. In *23rd Design Automation Conference, ACM/IEEE*, pages 314–319, July 1986.
- [14] E.R. Gansner, E. Koutsifos, S.C. North and K.P. Vo. A Technique for Drawing Directed Graphs. *IEEE Trans. Software Engg.*, 19:214–230, 1993.
- [15] G. Karypis, R. Aggarwal, V. Kumar, S. Sekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. 1997.
- [16] R. Kužnar and F. Brglez. PROP: A Recursive Paradigm for Area-Efficient and Performance Oriented Partitioning of Large FPGA Netlists. In *IEEE International Conference on Computer-Aided Design*, pages 644–649, November 1995.
- [17] M. Stallmann, F. Brglez, and D. Ghosh. Heuristics and Experimental Design for Bigraph Crossing Number Minimization. In *Proceedings of the First Workshop on Algorithm Engineering and Experimentation (ALENEX 99)*, January 1999. Also available at <http://www.cbl.ncsu.edu/publications/>.
- [18] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrt'ó. On bipartite drawings and the linear arrangement problem. In *Lecture Notes in Computer Science*, number 1467, pages 55–68. Springer Verlag, 1997.
- [19] Debabrata Ghosh. *Equivalence Classes for Experimental Design of Heuristics for Graph-Based NP-hard Problems*. PhD thesis, Electrical and Computer Engineering, North Carolina State University, Raleigh, N.C., May 2000. In progress.
- [20] F. Brglez and H. Fujiwara. Special Session on ATPG (Also introducing 'A Neutral Netlist of 10 Combinational Benchmark Circuits'). In *Int. Symp. On Circuits and Systems*, 1985. A basis for a benchmark directory ISCAS85, now archived at <http://www.cbl.ncsu.edu/benchmarks/>.
- [21] K. Kozminski, (Ed.). OASIS2.0 User's Guide. MCNC, Research Triangle Park, N.C. 27709, 1992. (Over 600 pages, distributed to over 60 teaching and research universities worldwide).
- [22] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *Proceedings of the 7th International Workshop on Field-Programmable Logic*, pages 213–222, August 1997. Software and postscript of paper can be downloaded from <http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html>.