VLSI DESIGN
1999, Vol. 10, No. 1, pp. 57–70
Reprints available directly from the publisher
Photocopying permitted by license only

Placement with Incomplete Data*

MAOGANG WANG[†], PRITHVIRAJ BANERJEE and MAJID SARRAFZADEH

Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208

(Received 7 September 1998; In final form 20 November 1998)

Traditional placement problems are studied under a fully specified cell library and a complete netlist. However, in the first, *e.g.*, 2 years of a 2–3 year microprocessor design cycle, the detailed netlist is unavailable. For area and performance estimation, layout must nevertheless be done with incomplete information. Another source of incompleteness comes from logic synthesis changes; some instances and their parameters will change as the project evolves. In the re-configurable computing area, sometimes we need to perform quick placement before all information is available. The problem of placement with incomplete data (PID) can be abstracted as having to place a circuit when p_c% of the cells and p_n% of the nets are missing. The key challenge in PID is how to add missing cells and nets.

In this paper, two "patching-methods" for adding missing nets and cells are proposed. The methods are called *abstraction* and *fusion*.

Experimental results are very interesting. First, they show that PID is a difficult problem and an arbitrary (and perhaps intuitively sound) method may not produce high-quality results. Experiments verify that the abstraction method is a very good predictor and that fusion is not. Specifically, when a circuit has 10% incompleteness, abstraction can predict the final total wirelength with an error of 5.8% while fusion has a 67.8% error in predicting the wirelength in the same circuit.

Keywords: Placement, physical layout, incomplete data, reconstruct, statistical

1. INTRODUCTION

The placement problem is defined as problems given a set of cells and a netlist of connections, optimizing the placement to minimize area, wirelength, delay, *etc*. In this problem, 100% cells and netlist is known.

Numerous placement algorithms exist, such as force directed approach [1, 7, 18, 6], simulated annealing [14, 15], and partitioning-based placement algorithms [3, 2, 5, 8]. A constructive placement method that employs resistive networks as a working domain was proposed in [4]. Various optimization objectives have been used. In [17], a comparison

*This work was supported in part by grant MIP-9527389 for NSF and grant DABT-63-97-C-0035 for DARPA.
[†]Corresponding author. Tel.: 847-491-9925, Fax: 847-461-4144, e-mail: mgwang@ece.nwu.edu

between linear and quadratic cost functions was reported. The linear cost function used in the GordianL placement tool achieves results with up to 20% less area than the quadratic cost function of the original Gordian procedure. Therefore, in this paper, we will use the half-perimeter cost function for area minimization.

The timing-driven placement problem has also been studied. The notion of zero-slack was introduced in [11]. In [9] criticality was used as a guide to select cells from a cell library. For a history of the timing-driven placement problem, see [10, 16, 13].

In this paper we study the placement problem with incomplete data (PID). This problem is of fundamental importance since VLSI design is getting more and more complicated. Now a state-of-the-art VLSI design contains millions of gates, and this number keeps increasing according to Moore's law. Such a complicated design will typically take 2 to 3 years to complete. In the first 2 years, a detailed netlist is not available. However, for area estimation and performance estimation, several placement runs are done with such incomplete information. Traditional placement problems are based on a fully specified cell library and a complete netlist. Obviously, this cannot handle this problem.

Another source of incompleteness will come from re-use of instances from earlier generations. The hardware instances are modified to some extent and more information may be added as the project moves forward. Thus even if we do have a complete cell library and netlist, this cell library and netlist will not be the same library and netlist as in the final version of the design. The designer will definitely want to know how much he or she can trust the placement result. If 5% of the cells in the library are changed later, it will not be good news to know the placement results will therefore change by 50%. In the re-configurable computing area, this problem is also practical. Sometimes we need to perform quick placement before all information is available from a compiler.

PID is the problem of trying to place the final design while only a part of the netlist and a fraction of the cell library is known. PID is of fundamental importance. It is also a very challenging problem, as will be shown later. To the best of our knowledge, no research has been done on this problem.

In our experiments, we use MCNC benchmarks. We randomly delete p_n% of the nets and p_c% of the cells. We then try to re-construct the netlist using different "patching-methods". We compare the wire-length of the original circuit against the wire-length of the "patched" circuit as a basis to judge the quality of the proposed patching-methods. Experiments on different values of p_c and p_n and different benchmarks have been performed.

This paper is organized as follows: In Section 2 we introduce some terminology which we are going to use throughout the paper. In Section 3 we formally described the *placement with incompleteness* problem followed by a detailed summary of approaches and implementations in Section 4. In Section 5, experimental results will be shown and discussed, followed by conclusions in Section 6.

2. TERMINOLOGY

In this paper we assume that we are given a *complete* synchronous circuit denoted $\mathcal{C}(\mathcal{M},\mathcal{N})$, which consists of a set of modules $\mathcal{M} = \{M_i | i = 1,\dots,|\mathcal{M}|\}$, and a set of nets $\mathcal{N} = \{N_i | i = 1,\dots,|\mathcal{N}|\}$. The set of modules consists of four subsets: primary inputs $\mathcal{I}$, primary outputs $\mathcal{O}$, storage elements $\mathcal{R}$, and combinational elements $\mathcal{A}$, *i.e.*, $\mathcal{M} = \mathcal{I} \cup \mathcal{O} \cup \mathcal{R} \cup \mathcal{A}$. A sample circuit is given in Figure 1.

Each *net* N_k consists of a set of terminals. For simplicity, it will be assumed that each net is driven by a single module output, or *source terminal* $s_{0,k}$,

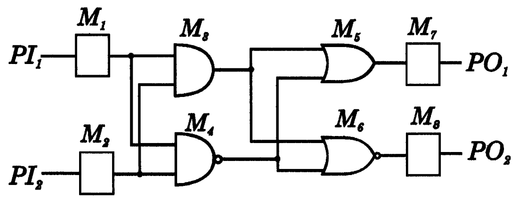

FIGURE 1 Sample circuit.

and that the remaining *sink terminals* $S_k - \{s_{0,k}\}$ are module inputs. This assumption simplifies the capture of signal flow directions and is used in the construction of the timing model. Terminals are given a location on the surface of an IC by the placement process. This location is used by many of the physical design steps. The location of a terminal is represented by $s_{i,k} = (x_{i,k}; y_{i,k})$, thus $S_k \subset \Re^2$, where $\Re$ is the set of real numbers and $s_{i,k}$ is the ith terminal in net net N_k. The rectilinear distance between two terminals $s_{i,k}$, $s_{j,m}$ is $\|s_{i,k}, s_{j,m}\| = |x_{i,k} - x_{j,m}| + |y_{i,k} - y_{j,m}|$. Similarly, each module M_j contains a set of terminals $S(M_j) = \{s_{j,k} | i = 0 \ldots |S(M_j)|\}$. Let the location on the plane of the module M_j be denoted by (x_{P_j}, y_{P_j}). Then the location of a terminal $s_{j,k} \in S(M_j)$ is a function of the module location.

For this work, we only consider standard cell layout style. We shall use the terms modules and cells interchangeably. Restriction on standard cells adds simplicity to our initial algorithms and implementation. The concepts developed here can be applied to other styles. However, we have not explored them yet.

3. PROBLEM FORMULATION

The general idea of the placement problem with incomplete data has been illustrated in Section 1. We are going to further formalize the problem here in this section.

During a design process, what we have in hand is the incomplete netlist and library. The complete netlist and library in the future will never be known at this time. Suppose there is an algorithm which can predict the final placement result only using the known incomplete information about the circuit. The question then is how good this result is. Therefore it is very important to first establish a standard for evaluating any PID algorithm.

Obviously, the best way to do this is to start from a complete circuit. This circuit, including the complete netlist and the cell library, will be no doubt the only fair metric to measure how good any kind of PID algorithm is. A good algorithm will give a good prediction on area or performance using only part of the original circuit information.

Starting from this complete circuit $\mathcal{C}$, we can easily build an incomplete circuit $\mathcal{C}_i$ simply by deleting or modifying cells and nets from the original circuit $\mathcal{C}$. There are two numbers to quantitively measure the incompleteness, p_c and p_n. We denote by p_c the percentage of cells we deleted or modified from $\mathcal{C}$ to get $\mathcal{C}_i$. Similarly, p_n is the percentage of nets we deleted or modified from $\mathcal{C}$ to get $\mathcal{C}_i$. For example, a complete circuit $\mathcal{C}$ contains 100 cells and 100 nets. After modification, the incomplete circuit has only 90 cells and 95 nets which are still the same as in $\mathcal{C}$. The other 10 cells and 5 nets are either deleted or modified. Then in this case, $p_c = 10\%$ and $p_n = 5\%$.

Now we are ready to describe the *placement with incompleteness* problem formally:

Given a complete circuit $\mathcal{C}(\mathcal{M}, \mathcal{N})$, an incomplete circuit $\mathcal{C}_i(\mathcal{M}_i, \mathcal{N}_i)$ is formed by deleting a number of cells and nets from $\mathcal{C}(\mathcal{M}, \mathcal{N})$. Two real numbers p_c and p_n satisfy the following conditions:

1. $0 < p_c < 1$, $0 < p_n < 1$.
2. $|\mathcal{M} \cap \mathcal{M}_i| = (1 - p_c) \cdot |\mathcal{M}|$
3. $|\mathcal{N} \cap \mathcal{N}_i| = (1 - p_n) \cdot |\mathcal{N}|$

We are going to make predictions on issues like final chip area, power consumption, *etc.*, of the circuit $\mathcal{C}(\mathcal{M}, \mathcal{N})$ based only on the circuit $\mathcal{C}_i(\mathcal{M}_i, \mathcal{N}_i)$ and parameters p_c, p_n. These predictions will be compared to the actual value of the circuit $\mathcal{C}(\mathcal{M}, \mathcal{N})$.

This above description concludes the formulation of the placement problem with incomplete data. There will be numerous ways to obtain such predictions for circuit $\mathcal{C}$. In this paper, we are actually trying to "reconstruct" the circuit $\mathcal{C}$ from the incomplete circuit $\mathcal{C}_i$. Specifically, we are going to find a set of modules $\mathcal{M}_a$ and a set of nets $\mathcal{N}_a$. We use $\mathcal{M}_a$ and $\mathcal{N}_a$, together with $\mathcal{M}_i$ and $\mathcal{N}_i$, to form a new circuit $\mathcal{C}'((\mathcal{M}_a \cup \mathcal{M}_i), (\mathcal{N}_a \cup \mathcal{N}_i))$.

Figure 2 shows an example of a complete circuit $\mathcal{C}$ with 8 cells and 10 nets. The incomplete circuit

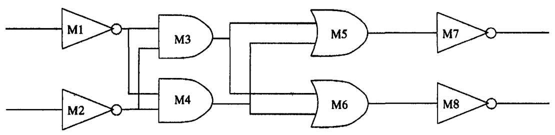

FIGURE 2 Original circuit C.

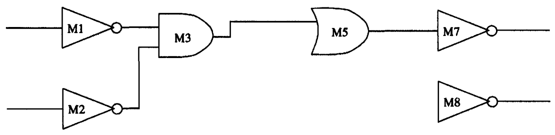

FIGURE 3 Incomplete circuit C_i.

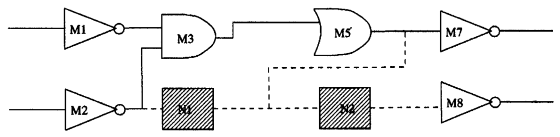

FIGURE 4 Reconstructed circuit C'.

C_i can be obtained by deleting cells $M4$, $M6$ and all the nets attached to them. Figure 3 shows the result of C_i. In this case, $p_c = 25\%$ and $p_n = 20\%$ since only 2 nets are gone. We can add two new dummy cells, $N1$ and $N2$ and some dummy nets to get circuit C'. Figure 4 shows a possible way to get C'. Of course, most probably, C' will not be exactly the same as the original circuit C. Our hope is that by using a "clever" way to get C', the placement result of C' will be very close to the placement result of C.

Now the question is how can we get the sets $\mathcal{M}_a$ and $\mathcal{N}_a$. One approach is by randomly determining all the details, such as the size of the new dummy cells, the number of pins on the new dummy cells, *etc*. Obviously, this is not a good method since we have not made use of the known information. Since the missing part of the circuit will not be too large, we will expect the missing part to be "very much" like the rest of the circuit.

In the next section, we will study appropriate approaches to this problem.

Note that in this paper, we only try to make a prediction on the total net lengths, which is consistant with the final chip area. "Half perimeter bounding box" is what we will use to measure the length of each net.

4. OUR APPROACH AND IMPLEMENTATION

As mentioned in the previous section, a totally random approach is not good. On the other hand, we should notice the following fact:

FACT 1 *There are no algorithms which can always reconstruct a totally identical circuit as circuit C using only the imcomplete circuit C_i.*

Proof The proof is simple. Suppose C_i is the incomplete circuit with which we will start with. Then arbitrarily add two different sets of cells and nets to C_i. This will give us two different circuits C_1 and C_2, and they share the same incomplete circuit C_i. After we run our "universal" algorithm on C_i, we get the reconstructed circuit C'. If $C' = C_1$, then $C' \neq C_2$, and *vice versa*. This means that this "universal" algorithm cannot get an identical circuit for both C_1 and C_2 at the same time. Since both C_1 and C_2 are the correct results, the above fact is true. ■

This fact suggests that the problem itself is very hard since we have no hope of getting a completely correct answer. Before we start thinking about the actual approach, let us first make some assumptions about this problem.

First, the incompleteness measurement p_c and p_n should not be too large. We cannot get any sort of prediction if we only know 5% of the netlist. We should know at least half of the original circuit, and it is very reasonable to assume that the missing part of the circuit is very much like the remaining part of it. That is, if we miss a very special part in this incomplete circuit, we should not expect to get a decent prediction.

Following this assumption, sets $\mathcal{M}_a$ and $\mathcal{N}_a$ should be "similar" to sets $\mathcal{M}_i$ and $\mathcal{N}_i$ in circuit C_i. The question is, what does "similar" mean here. Our interpretation for this is that they have similar statistics. For example, two module sets should have similar size distribution curves (to be discussed later).

There are a lot of possible statistics on which we can base our studies. Among these, we chose three of them to be the statistics for the module set. They are:

1. The size distribution of cells.
2. The number of terminals distribution on each cell.
3. The I/O type distribution of all terminals.

There is no question that the size of cells is the most important information in area estimation. Besides this, the number of terminals on each cell is also important since it determines the connectivity of each cell. The larger this number is, the larger the total final area will be. An appropriate I/O type distribution of terminals will assure a netlist similar to the original circuit. An I/O type distribution means that we should know approximately how many terminals we are going to add are inputs, how many are outputs and how many are bidirectional. Without this I/O type distribution, any pair of terminals can be hooked up, leaving too much freedom to deal with. If we know this distribution, since an output terminal has to be hooked up with an input or bidirectional one, it is easier for us to construct a "similar" netlist as the original one.

The above is a brief description of these statistics and reasons why they are important to us. We will discuss them in detail in the following paragraphs.

The size distribution function $P_s(s)$ is a probability function of cell size. For any possible physical size s of a cell, $P_s(s)$ will give us the probability that a cell in the circuit has size s.

Since the probability function $P_s(s)$ is not analytical, we will use a linear array to represent it. Thus we need to quantitize the size s first.

The actual implementation is: first find the minimum cell size $s_{\min}$ and the maximum cell size $s_{\max}$ in the circuit. We assume that function $P_s(s)$ will have a non-zero value only when $s_{\min} \leq \text{size} \leq s_{\max}$. This means, all the cells to be added later will have sizes no larger than $s_{\max}$ and no smaller than $s_{\min}$.

Then we evenly divide the range $[s_{\min}, s_{\max}]$ into 100 sub-ranges so that we have 100 buckets. Each bucket corresponds to a sub-range and records the number of cells whose size falls into this sub-range. Finally, in order to get a real probability function, we normalize the values in all buckets. The result is the size distribution function which actually is a discrete function.

Let us revisit the example shown in Figures 2, 3 and 4. In that circuit, we have 3 types of cells, inverters, AND gates and OR gates. Suppose an inverter has a size of 1, and AND gate and an OR gate each have a size of 100. Note that all statistics should be derived from circit C_i since C_i is the only information we know in PID. As shown in Figure 3, we have four inverters, one AND gate and one OR gate. Thus the number in our first bucket is 4, the number in our last bucket is 2, and the numbers in the other buckets are all zero. After normalization, the probability of being in the first bucket is 66.7%, the probability of being in 100th bucket is 33.3% and probability of being in other buckets is zero.

Figure 5 shows this size distribution function P_s.

The probability function P_t is a function of the number of terminals in a cell. It tells us for a certain cell, what the probability is of having a certain number of terminals on it. Obviously, this

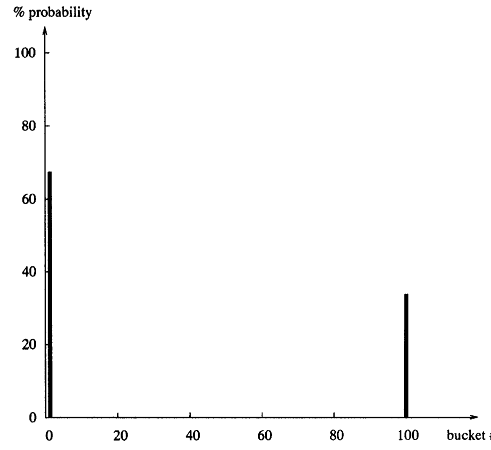

FIGURE 5 Size distribution function P_s for circuit in Figure 3.

function is not independent of the size distribution. A smaller cell will almost certainly have fewer terminals than a larger cell. We have to take this fact into account. Remember we will have 100 buckets for cell sizes. Then for each bucket, we have a separate distribution function of the number of terminals in a cell.

Take the same example circuit C_i as above. The first bucket contains only inverters, so in this bucket, all the cells have a 100% chance to have 2 terminals. The last bucket contains AND gates and OR gates, but since they both have 3 terminals, so all the cells there have a 100% chance to have 3 terminals. Figure 6 shows two distribution functions of the number of terminals corresponding to the first and the last bucket.

Finally, due to the same reason as above, the I/O type distribution for terminals is not an independent function either. Thus we have a separate I/O type distribution function for each bucket. Specifically, for all cells in a certain bucket, count the number of input, output and bidirectional terminals. Then normalize the values. Again for circuit C_i shown in Figure 3, the first bucket has four inverters, so 50% of the terminals are inputs and 50% of them are outputs. Similarly, the last bucket has one AND gate and one OR gate, so 66.7% of the terminals are inputs and 33.3% of

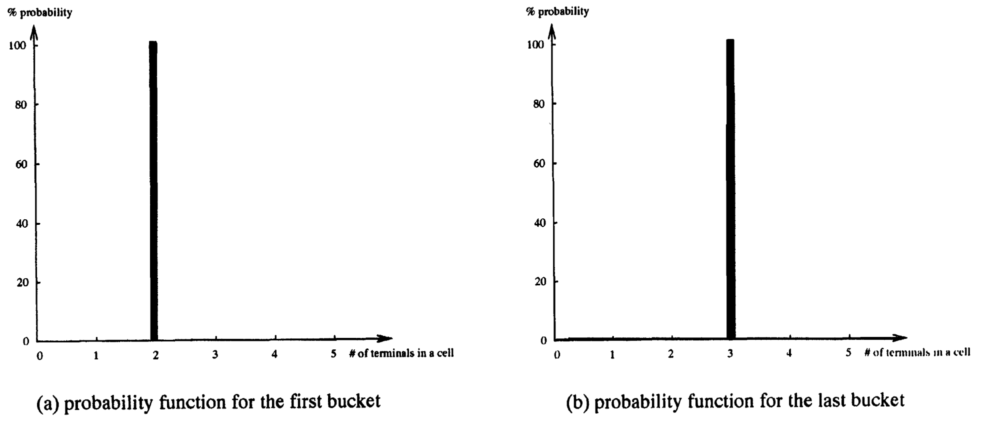

FIGURE 6 Distribution function P_t for circuit in Figure 3.

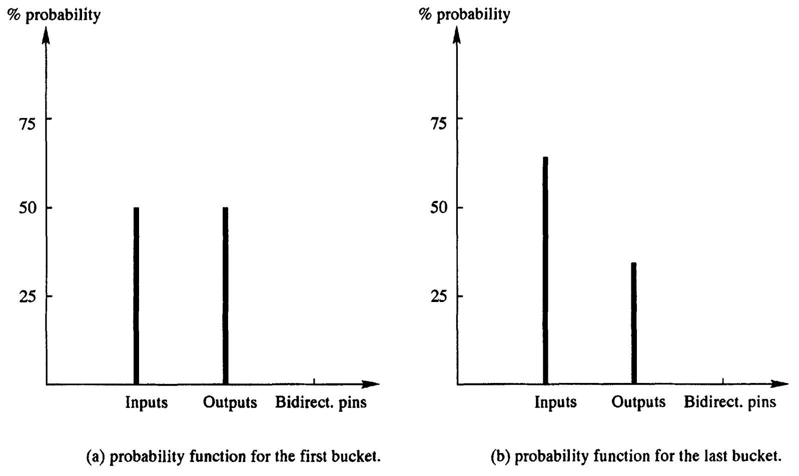

FIGURE 7 I/O type distribution function for circuit in Figure 3.

the terminals are outputs. Figure 7 shows two I/O type distribution functions for these two buckets.

This is useful in determining the I/O type of a newly added terminal. If a terminal belongs to a cell in bucket 1, there is a 50% chance that it will be an input or output terminal.

It is natural to have these three for the set of cells. If timing information is an issue in the placement, we should also include some statistics about the timing information. For constructing a missing cell, we think these three statistics are adequate to represent the characteristics of the set $\mathcal{M}_i$.

Based on this discussions, we can write the pseudo-code for finding missing cells, set $\mathcal{M}_a$ as following:

```
1. AddCell(circuit $\mathcal{C}_i, p_c$) {
2.   Get these three statistics discussed above.
3.   $n_c = (p_c \cdot |\mathcal{M}_i|/1 - p_c)$
4.   for $i = 1$ to $n_c$
5.     Create a cell data structure.
6.     Determine the size of this cell using the size
       distribution.
7.     Determine the number of terminals on
       this cell using the appropriate distribution
       function.
8.     For each terminal on the cell, determine its
       type using the appropriate terminal type
       distribution.
9. }
```

In Steps 6, 7 and 8, we use randomly generated numbers and distribution functions to determine different properties of each missing cell. For example, the size distribution function $P_s(s)$ gives the percentage of cells in each of the 100 buckets. Thus we can randomly pick a bucket according to this probability function. The size of the cells in this bucket will be assigned to the to be added missing cell. This method can also apply to Steps 7 and 8.

After this procedure is done, we have added a set of cells $\mathcal{M}_a$ to the circuit $\mathcal{C}'$. These newly added cells are isolated since no nets are connected to them. The next step is to add some nets to the incomplete circuit $\mathcal{C}_i$. We will refer to all the existing nets already in $\mathcal{C}_i$ as "old nets" and all the nets we will add later as "new nets". Similarly, all the cells previously existing in $\mathcal{C}_i$ are called "old cells" and all the cells added by AddCell( ) are called "new cells".

Things are not so easy when finding the set of new nets, $\mathcal{N}_a$. The reason for this is because the set of nets is more like a graph, so it is vague to say one graph is more similar to a given graph than another one. On the other hand, netlist information is the very factor which we cannot ignore.

In the actual implementation, we use only one statistic for the set of nets, the number of terminals in each net. That is, we need to know how many nets are 2-pin nets, how many nets are 3-pin nets, etc. This is no doubt be an important statistic, but it should not be the only one to determine the set of nets. The reason why we only chose this one is because:

1. It is not clear what other statistics will be good representatives for the set $\mathcal{N}_i$.
2. This one is easy to implement, and other useful statistics can be easily added later.

Even after deciding to use only one statistic, it is still not clear how to get the whole missing netlist, the set $\mathcal{N}_a$.

The first thing we need to know is the number of nets to be added. This can be found by using the input parameter p_n.

After that, a natural scheme will be similar to what we did on adding new cells: Look at all the cells in the circuit, including those "old" cells in $\mathcal{C}_i$ and those "new" cells added by AddCell( ), and create a list $\mathcal{L}$ of all the terminals unattached to any nets. Initially, this list $\mathcal{L}$ will include all terminals from all new cells and those unattached terminals in the old cells.

As the example shown in Figure 4, this list $\mathcal{L}$ will contain two pins in $N1$, two pins in $N2$ and the input pin in $M8$.

After the list $\mathcal{L}$ is created, for each to be added new net, first determine the number of terminals in it using the distribution function. Then assign terminals from the list $\mathcal{L}$ to this new net and delete those assigned terminals from the list $\mathcal{L}$. Repeat this procedure on the next to be added new net.

One of these following three cases will happen during this procedure.

1. List $\mathcal{L}$ is empty after all nets have been added. (This is great.) Then we have finished our reconstruction procedure on the circuit $\mathcal{C}'$.
2. There are no terminals left in list $\mathcal{L}$ when adding a new net.
   Then we can randomly pick any terminals in the circuit $\mathcal{C}'$ and assign it to this net.
3. There are still some terminals left in list $\mathcal{L}$ after all nets have been added.

Actually, Case 3 will happen a lot in real life. This is because of the way we get the circuit $\mathcal{C}_i$. For example, if in the original circuit $\mathcal{C}$, a 4-pin-net is connected to cell A, B, C and D, and later we only delete cell D from the circuit $\mathcal{C}$ to form the circuit $\mathcal{C}_i$ while retaining cells A, B and C, then this net will still exist in the incomplete circuit $\mathcal{C}_i$, except that now it becomes a 3-pin-net instead of a 4-pin-net.

This fact suggests that some terminals in newly added cells should be assigned to an old net as well. That is why the number of terminals in list $\mathcal{L}$ is almost always larger than the total number of terminals in all new nets.

Based on this analysis, if Case 3 happens, for each terminal left in the list $\mathcal{L}$, we will randomly pick either an old net or a new net and assign this terminal to it. We will refer to this patch method as *fusion* as shown in Figure 8.

However, *fusion* is not the only way to accomplish such a task. We have an alternative patching method, *abstraction*, described as the following: if Case 3 happens, instead of assigning the remaining terminals to an existing net as in *fusion*, we will keep adding new nets and assign these terminals to these new nets until the list $\mathcal{L}$ becomes empty. Since most terminals initially in list $\mathcal{L}$ are just terminals in new cells, in *abstraction*, all new cells are inter-connected and rarely is there a connection between new cells and old cells. This is shown is Figure 9.

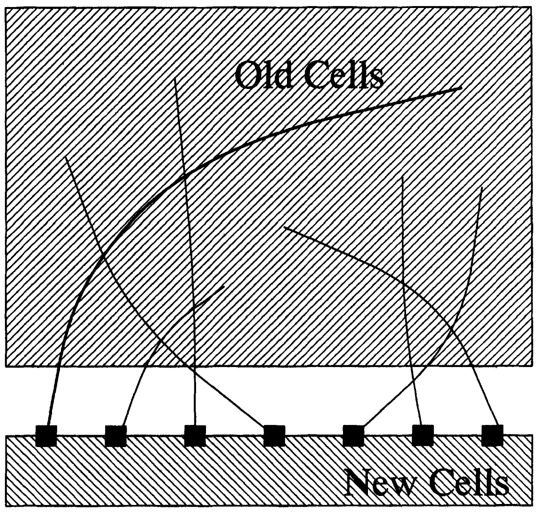

FIGURE 8 Method *fusion*.

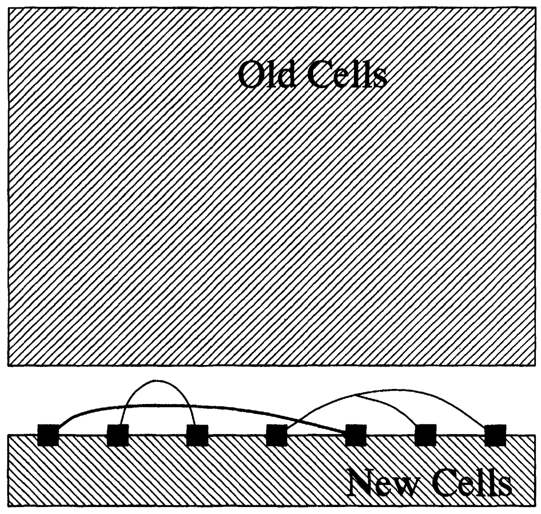

FIGURE 9 Method *abstraction*.

We briefly summarize the difference between *fusion* and *abstraction* as in the following: In *fusion*, new cells are attached to both new nets and old nets, so they are connected to both old cells and new cells; while in *abstraction*, new cells are only attached to new nets, new cells form kind of a cluster since most of their nets are internal nets. Another difference is that the circuit $\mathcal{C}$ created by *abstraction* will have more nets than its counterpart created by *fusion*.

Based on these discussions, we write the pseudo-code for both patching methods as in following:
Fusion:

```
1. AddNetFusion(circuit $\mathcal{C}_i, p_n$) {
2.   Get net statistics discussed above.
3.   Create unassigned terminal list $\mathcal{L}$.
4.   $n_n = p_n \cdot |\mathcal{N}_i|$
5.   for $i = 1$ to $n_n$
6.     Create a net data structure.
7.     Determine the number of terminals on this
       net using the appropriate statistics.
8.     Randomly pick $n_n$ terminals from $\mathcal{L}$.
9.     Assign these $n_n$ terminals to this net.
10.    Delete these $n_n$ terminals from $\mathcal{L}$.
11.  while $\mathcal{L}$ is not empty, do {
12.    Get a terminal from $\mathcal{L}$.
13.    Randomly pick a net from $\mathcal{N}_i$.
14.    Assign this terminal to this net.
15.    Delete this terminal from $\mathcal{L}$.
16.  }
17. }
```

Abstraction

```
1. AddNetAbstraction(circuit $\mathcal{C}_i$) {
2.   Get net statistics discussed above.
3.   Create unassigned terminal list $\mathcal{L}$.
4.   while $\mathcal{L}$ is not empty, do {
5.     Create a net data structure.
6.     Determine the number of terminals on this
       net using the appropriate statistics.
7.     Randomly pick $n_n$ terminals from $\mathcal{L}$.
8.     Assign these $n_n$ terminals to this net.
9.     Delete these $n_n$ terminals from $\mathcal{L}$.
10.  }
11. }
```

Intuitively, *fusion* makes much more sense than *abstraction*, since it appears that *fusion* will generate a circuit more like the original one, circuit $\mathcal{C}$, while *abstraction* will almost never look like $\mathcal{C}$.

Contrary to people's intuition, experimental results show that *abstraction* offers much more accurate prediction than *fusion* does. In the next section, we will present this interesting experimental result, followed by an explanation.

5. EXPERIMENTAL RESULTS AND DISCUSSION

In Section 4, we gave the pseudo-code for our approaches. These approaches were implemented using C++. All experiments were run on a Sun Sparc-20 workstation. Three MCNC benchmark standard cell circuits were used for experiments. Table I shows some basic statistics for these three benchmark circuits. We delete all the I/O pads from all three of these circuits, allowing us to concentrate on the effects of incomplete data on core cells. As shown in Table I, circuit Nprimary1 is the MCNC benchmark circuit primary1 without I/O pads, and circuit Nprimary2 and Nstruct are MCNC benchmark primary2 and struct without I/O pads, respectively.

We feed the reconstructed circuit to the high quality placement tool called NRG [12]. Since this is a statistical approach, we needed to run a good number of trials to get the data. Thus we run NRG using the fast mode.

For each benchmark circuit $\mathcal{C}(\mathcal{M},\mathcal{N})$, for a specified degree of incompleteness p_c. We first delete $p_c \cdot |\mathcal{M}|$ cells and the nets attached to them to form $\mathcal{C}_i$. Then we run our two reconstruction procedures, fusion and abstraction, to get a "reconstructed" circuit $\mathcal{C}'$ (*fusion and abstraction* were discussed in detail in Section 4). After that, we use the NRG placement tool to get the total wirelength of $\mathcal{C}'$ and compare this value with the value of the original circuit $\mathcal{C}$.

Table II shows the basic statistics for the "reconstructed" circuit *Nprimary*1′ using different degrees of incompleteness. The first row, 0% incompleteness, is the statistics for the original circuit Nprimary1′.

Table III shows the total wirelength results achieved from the reconstructed circuit *Nprimary*1′ For each degree of incompleteness, column "Ave WL" shows the average total wirelength over all trials; column "std. Dev." shows the standard deviation of all trials, and "% error" shows the percent error compare to the real value from the original complete circuit Nprimary1.

TABLE I Statistics for tested benchmark circuits

Circuit	# cells	# I/O pads	# pins	# nets	# rows	WL
Nprimary 1	750	0	3062	1156	16	642853
Nprimary 2	2907	0	11681	3671	32	3215512
Nstruct	1888	0	5407	1920	16	374940

TABLE II Netlist statistics for Nprimary1 after reconstructing the circuit

	Fusion				Abstraction			
% of incomp.	# trials	# cells	# pins	# nets	# trials	# cells	# pins	# nets
0%	–	750	3062	1156	–	750	3062	1156
5%	10	750	3049	1156	10	750	3052	1198
10%	10	750	3071	1156	10	750	3072	1240
15%	10	750	3065	1156	10	750	3064	1270
20%	10	750	3094	1156	10	750	3092	1314
25%	10	750	3109	1156	10	750	3106	1362
30%	10	750	3061	1156	10	750	3074	1376

TABLE III Wirelength results for Nprimary1 after reconstructing the circuit

	Fusion				Abstraction			
% of incomp.	# trials	Ave WL	std. Dev.	% error	# trials	Ave WL	std. Dev.	% error
0%	10	642853	6782	0%	10	642853	6782	0%
5%	10	740452	11697	15.2%	10	609263	9470	– 5.2%
10%	10	874490	14284	36.0%	10	605948	6398	– 5.7%
15%	10	941024	12486	43.4%	10	593153	7697	– 7.7%
20%	10	1080158	16305	68.0%	10	597267	4250	– 7.1%
25%	10	1202426	10309	87.1%	10	599750	8384	– 6.7%
30%	10	1231592	11368	91.6%	10	560518	9623	– 12.8%

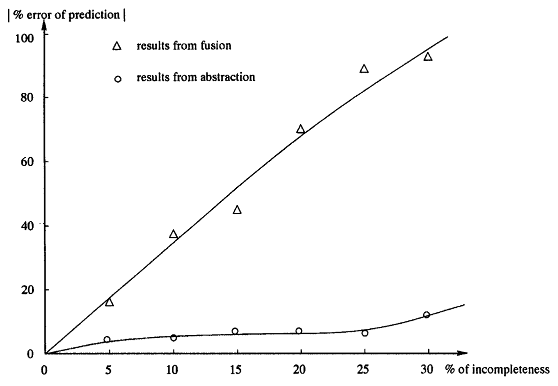

FIGURE 10 Prediction error *vs.* percentage of incompleteness for Nprimary1.

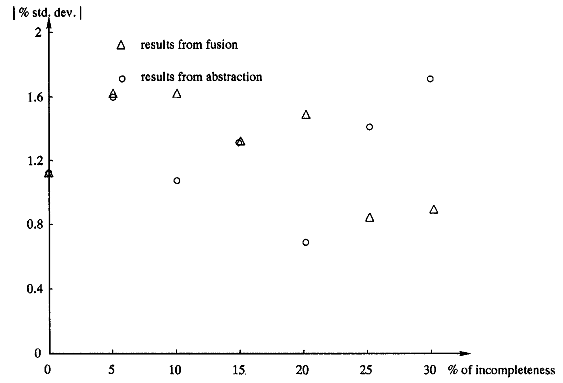

FIGURE 11 Std. deviation *vs.* percentage of incompleteness for Nprimary1.

Figure 10 shows the prediction error *vs.* percentage of incompleteness for circuit Nprimary1. It's clear that *fusion* has a much bigger error than *abstraction* for any percentage of incompleteness.

Since this is a non-deterministic procedure, standard deviation is another important factor to look at. Figure 11 shows the std. deviation *vs.* percentage of incompleteness for Nprimary1. However, there is not a clear pattern in this figure. All the data points are spread in the range between 0.7% and 1.7%. Standard deviation does not increase as the percentage of incompleteness increases. This is determined by the nature of *fusion* and *abstraction*. Since *fusion* and *abstraction* rely heavily on the statistics from the incomplete circuit C_i, the increase of incompleteness really does not affect the degree of randomness in reconstructing circuit C'. That is why the standard deviation does not increase along with the percentage of incompleteness.

Similar statistics and results for Nprimary2 and struct were provided in Tables IV, V, VI and VII. Figure 12 shows the prediction error *vs.* percentage of incompleteness for circuit Nprimary2. Figure 13 shows the prediction error *vs.* percentage of incompleteness for circuit Nstruct.

Table VIII and Figure 14 are a summary table and a summary figure of these three circuits.

From Figures 10, 12 ,13 and 14 we can conclude that *abstraction's* prediction is much more accurate than *fusion's*. That is very surprising and counter-intuitive.

The reason we might think *fusion* is better than *abstraction* is because it seems that *fusion* has at least some chance to reconstruct the original circuit. However, the chances of perfectly

TABLE IV Netlist statistics for Nprimary2 after reconstructing the circuit

	Fusion				Abstraction			
% of incomp.	# trials	# cells	# pins	# nets	# trials	# cells	# pins	# nets
0%	–	2907	11681	3671	–	2907	11681	3671
5%	10	2907	11691	3671	10	2907	11689	3895
10%	10	2907	11684	3671	10	2907	11687	3998
15%	10	2907	11596	3671	10	2907	11597	4235
20%	10	2907	11663	3671	10	2907	11679	4395
25%	10	2907	11686	3671	10	2907	11692	4538
30%	10	2907	11735	3671	10	2907	11729	4646

TABLE V Wirelength results for Nprimary2 after reconstructing the circuit

	Fusion				Abstraction			
% of incomp.	# trials	Ave WL	std. Dev.	% error	# trials	Ave WL	std. Dev.	% error
0%	10	3215512	33806	0%	10	3215512	33806	0%
5%	10	5229648	55474	62.7%	10	3384501	87486	5.3%
10%	10	5772115	109263	79.5%	10	3418518	80982	16.3%
15%	10	7217765	92149	124%	10	3612617	142242	12.3%
20%	10	8575307	116364	167%	10	3576452	96928	11.2%
25%	10	9466555	90891	194%	10	3698880	93940	15.0%
30%	10	10029639	91762	212%	10	3836407	134678	19.3%

TABLE VI Netlist statistics for Nstruct after reconstructing the circuit

	Fusion				Abstraction			
% of incomp.	# trials	# cells	# pins	# nets	# trials	# cells	# pins	# nets
0%	–	1888	5407	1920	–	1888	5407	1920
5%	10	1888	5412	1920	10	1888	5408	2076
10%	10	1888	5408	1920	10	1888	5410	2186
15%	10	1888	5411	1920	10	1888	5410	2290
20%	20	1888	5389	1920	20	1888	5387	2375
25%	21	1888	5411	1920	20	1888	5408	2459
30%	29	1888	5399	1920	21	1888	5403	2565

TABLE VII Wirelength results for Nstruct after reconstructing the circuit

	Fusion				Abstraction			
% of incomp.	# trials	Ave WL	std. Dev.	% error	# trials	Ave WL	std. Dev.	% error
0%	10	374940	6456	0%	10	374940	6456	0%
5%	10	576419	5564	53.7%	10	360179	3976	−3.9%
10%	10	703867	4643	87.7%	10	354738	5807	−5.4%
15%	10	778197	4304	108%	10	354167	5796	−5.5%
20%	20	819950	2500	119%	20	340646	2787	−9.1%
25%	21	907997	4628	142%	20	353466	3556	−5.7%
30%	29	983995	3562	162%	21	315854	4331	−15.8%

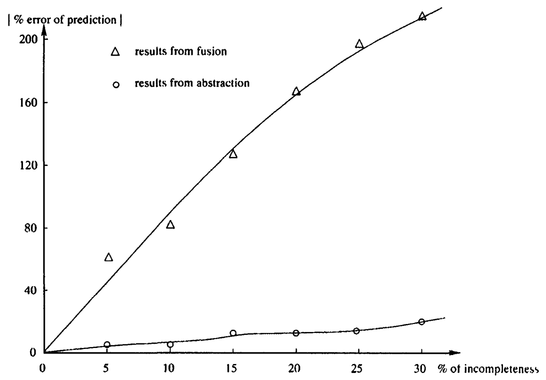

FIGURE 12 Prediction error *vs*. percentage of incompleteness for Nprimary2.

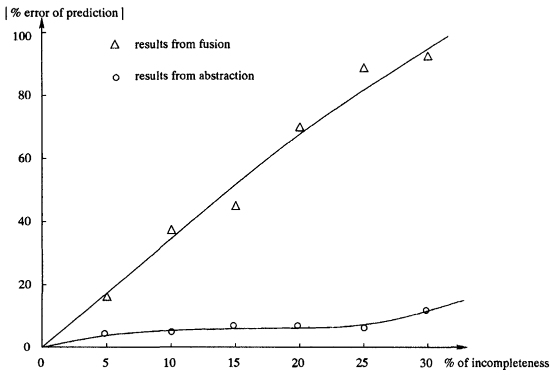

FIGURE 13 Prediction error *vs*. percentage of incompleteness for Nstruct.

TABLE VIII Average wirelength prediction results for all three testing circuits

% of incomp.	Absolute % error for fusion	Absolute % for error for abstraction
0%	0%	0%
5%	43.9%	4.8%
10%	67.8%	5.8%
15%	91.8%	8.5%
20%	118%	9.1%
25%	141%	9.1%
30%	155%	16.0%

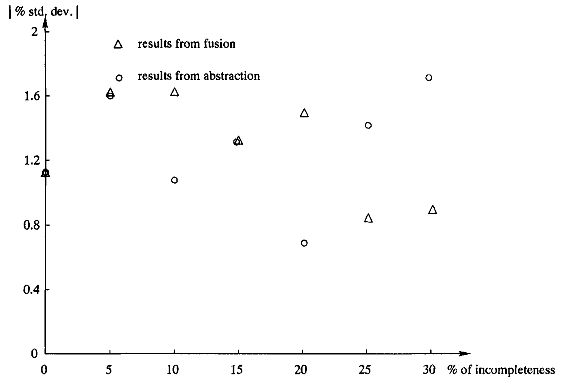

FIGURE 14 Prediction error *vs*. percentage of incompleteness for all three testing circuits.

reconstructing the original circuit is too small to be realized. According to the Fact described in Section 3, there is no hope to reconstruct an identical circuit as $\mathcal{C}$. Our only hope is that the "netlist-space" is smooth enough so that even if we make some mistakes in the wiring, the final total wirelength would not be affected by much. This is not a bad guess, but unfortunately, it is proven not to be true. Figures 10, 12, 13 show that with *fusion*, even for 5% incompleteness, the final result can be altered by more than 50% (Nprimary2 and Nstruct), which is not tolerable. The reason why *abstraction* is successful is because almost all the nets it adds are within the newly added cells. That is to say, it does not alter the netlist topology in the incomplete circuit C_i. As for the final prediction, the wirelength for the newly added part may not be correct, but at least the wirelength for the old part is very accurate. Based on the assumption we made in Section 3, the old part should be the dominating part, so we are going to get a decent prediction for the final result.

6. CONCLUSIONS AND FUTURE WORK

We formally proposed a new problem in placement area, placement with incomplete data. This problem arises along with the increasing complexity of current VLSI designs. It will become more and more important in the future.

We tried two approaches: *fusion* and *abstraction*. We want to see how good they can predict the final wirelength/chip area using only an incomplete netlist and library.

Experimental results show that *abstraction* is a much better patching method than *fusion*. With 10% incompleteness in the circuit, *abstraction* can still predict the wirelength with an error of 5.8%; while *fusion* can have an error more than 50% even with a 5% incompleteness in the circuit.

Comparisons between *fusion* and *abstraction* shows that the netlist topology is essential to the final wirelength. In order to get a good prediction, we should try to preserve the original topology as much as possible.

Although *abstraction* did a decent job on prediction, there are still quite a few things we need to work on in the future. If we look at Tables III, V and VII carefully, we will notice that for the same circuit, errors for *abstraction* are either all positive or negative. This suggests that some systematic error exists. This gives us an opportunity to find a way to make the prediction error even smaller. Based on some properties of circuit C_i, if we can predict the sign of the error, then we can add or substract a small amount from our prediction to get a more accurate value. However, the relationship between the signs of the predicted error and circuit properties is not known yet.

References

[1] Anway, H., Farnham, G. and Reid, R. (1985). "Plint Layout System for VLSI Chip Design". In: *Design Automation Conference*, pp. 449–452. IEEE/ACM.

[2] Breuer, M. A. (1997). "A Class of Min-cut Placement Algorithms". In: *Design Automation Conference*, pp. 284–290. IEEE/ACM.

[3] Breuer, M. A. (1977). "Min-cut Placement". *J. Design Automation and Fault-Tolerant Computing*, **1**(4), 343–382.

[4] Cheng, C. K. and Kuh, E. S. (1984). "Module Placement Based on Resistive Network Optimization". *IEEE Transactions on Computer Aided Design*, **3**(3), 218–225.

[5] Dunlop, A. E. and Kernighan, B. W., "A Procedure for Placement of Standard Cell VLSI Circuits". *IEEE Transactions on Computer Aided Design*, **4**(1), 92–98, January 1985.

[6] Eisenmann, H. and Johannes, F. M. (1998). "Generic Global Placement and Floorplanning". In: *Design Automation Conference*, pp. 269–274. IEEE/ACM.

[7] Hsu, C. P. *et al.* (1985). "APLS2: A Standard Cell Layout System for Double-layer Metal Technology". In: *Design Automation Conference*, pp. 443–448. IEEE/ACM.

[8] Huang, D. and Kahng, A. B., "Partitioning-based Standard-cell Global Placement with an Exact Objective". In: *International Symposium on Physical Design*, pp. 18–25. ACM, April 1997.

[9] Lin, S. P., Marek-Sadowska, M. and Kuh, E. S. (1990). "Delay and Area Optimization in Standard Cell Design". In: *Design Automation Conference*, pp. 349–352. IEEE/ACM.

[10] Marek-Sadowska, M., "Issues in Timing Driven Layout". *Algorithmic Issues in* VLSI Layout, pp. 1–24, 1993. Sarrafzadeh, M. and Lee, D. T., Editors.

[11] Nair, R., Berman, C. L., Hauge, P. S. and Yoffa, E. J., "Generation of Performance Constraints for Layout". *IEEE Transactions on Computer Aided Design*, **CAD-8**(8), 860–874, August 1989.

[12] Sarrafzadeh, M. and Wang, M., "NRG: Global and Detailed Placement". In: *International Conference on Computer-Aided Design. IEEE*, November 1997.

[13] Sarrafzadeh, M. and Wong, C. K., An *Introduction to VLSI Physical Design*, McGrwa Hill, 1996.

[14] Sechen, C. (1988). *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer, B., Deventer, V., The Netherlands, 1988.

[15] Sechen, C. and Sangiovanni-Vincentelli, A. (1986). "TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package". In: *Design Automation Conference*, pp. 432–439. IEEE/ACM.

[16] Sherwani, N. A. (1993). *Algorithms For VLSI Physical Design Automation*. Kluwer Academic Publishers.

[17] Sigl, G., Doll, K. and Johannes, F. M. (1991). "Analytical Placement: A Linear or a Quadratic Objective Function". In: *Design Automation Conference*, pp. 427–431. IEEE/ACM.

[18] Ohtsuki, T., Sudo, T. and Goto, S. (1983). "CAD Systems for VLSI in Japan". In: *Information and Control*, Vol. 59.

Authors' Biographies

Maogang Wang received his B.S. degree in 1994 from the University of Science and Technology of China at Hefei, China. He received his M.S. degree in Physics and Computer Engineering in 1996 and 1998 respectively from Northwestern University. Since 1996 he has been working with professor Majid Sarrafzadeh in the department of Computer Engineering at Northwestern University. His research interests lie in the area of physical layout in VLSI CAD. He is expecting his Ph.D. degree at April, 2000.

Prith Banerjee received his B. Tech. degree in Electronics and Electrical Engineering from the Indian Institute of Technology, Kharagpur, India, in August 1981, and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Illinois at Urbana-Champaign in December 1982 and December 1984 respectively. Dr. Banerjee is currently the Walter P. Murphy Chaired Professor of Electrical and Computer Engineering and Director of the center for Parallel and Distributed Computing at Northwestern University in Evanston, Illinois. Prior to that he was the Director of the Computational Science and Engineering program, and Professor of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. Dr. Banerjee's research interests are in Parallel Algorithms for VLSI Design Automation, and Compilers for Distributed Memory Parallel Compilers, and is the author of over 200 papers in these areas.

Majid Sarrafzadeh received his B.S., M.S. and Ph.D. in 1982, 1984, and 1987 respectively from the University of Illinois at Urbana-Champaign in Electrical and Computer Engineering. He joined Northwestern University as an Assistant Professor in 1987. Since 1997 he has been a Professor of Electrical Engineering and Computer Science at Northwestern University. His research interests lie in the area of VLSI CAD, design and analysis of algorithms and VLSI architecture. Dr. Sarrafzadeh is a Fellow of IEEE for his contribution to "Theory and Practice of VLSI Design". He received an NSF Engineering Initiation award, two distinguished paper awards in ICCAD, and the best paper award for physical design in DAC for his work in the area of High-Speed VLSI Clock Design. He has served on the technical program committee of numerous conferences in the area of VLSI Design and CAD, including ICCAD, EDAC and ISCAS. He has served as committee chairs of a number of these conferences, including International Conference on CAD and International Symposium on Physical Design.