Potential_NRG: Placement with Incomplete Data

Maogang Wang, Prithviraj Banerjee and Majid Sarrafzadeh
Department of Electrical and Computer Engineering
Northwestern University, Evanston, IL 60208

Abstract

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 reuse of instances from earlier design generations; these instances and their parameters will change as the project evolves. 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 and illurstrative. 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 because circuits produced by abstraction attain much of the properties of the origianl circuits. Summary Table 3 in Section 4 shows that 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.

1 Introduction

The placement problem has been an interesting and challenging problem for many years. Recently, performance-driven, issues related to deep submicron, reliability and many more factors make the placement problem even more challenging.

Numerous approaches to the placement problem have been proposed, such as force directed [1, 6], simulated annealing [8], and partitioning-based placement [3, 2, 5]. A constructive placement method that employs resistive networks as a working domain was proposed in [4]. In [9], there is a comparison between linear and quadratic cost functions. The linear cost function used in the Gordian**L** 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.

Here we study the placement problem with incomplete data (PID). This problem is of fundamental importance since VLSI design is getting more and more complicated. A typical VLSI design will 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 reuse of instances from earlier generations. The hardware instances are modified to some extent and more information may be added as the project moves forward. So 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 surely want to know how much he or she can trust the placement result. 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.

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 formally described the *placement with incompleteness* problem followed by a detailed summary of approaches and implementations in Section 3. In Section 4, experimental results will be shown and discussed, followed by conclusions in Section 5.

2 Problem Formulation

The best way is to study PID problem is starting 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 a complete circuit $\mathcal{C}$, we can 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 $\mathcal{C}_i$. 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 1 shows an example of a complete circuit $\mathcal{C}$ with 8 cells and 10 nets. The incomplete circuit $\mathcal{C}_i$ can be obtained by deleting cells $M4$, $M6$ and all the nets attached to them. Figure 2 shows the result of $\mathcal{C}_i$. 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 $\mathcal{C}'$. Figure 3 shows a possible way to get $\mathcal{C}'$. Of course, most probably, $\mathcal{C}'$ will not be exactly the same as the original circuit $\mathcal{C}$. Our hope is that by using a "clever" way to get $\mathcal{C}'$, the placement result of $\mathcal{C}'$ will be very close to the placement result of $\mathcal{C}$.

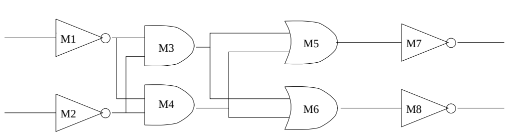

Figure 1: Original circuit $\mathcal{C}$.

In the next section, we will show our approaches to find $\mathcal{M}_a$ and $\mathcal{N}_a$.

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.

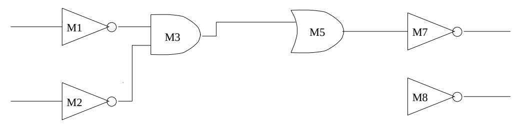

Figure 2: Incomplete circuit $\mathcal{C}_i$.

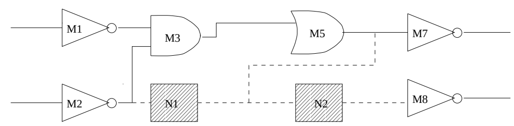

Figure 3: Reconstructed circuit $\mathcal{C}'$.

3 Our Approach and Implementation

Following the discussion in the previous section, sets $\mathcal{M}_a$ and $\mathcal{N}_a$ should be "similar" to sets $\mathcal{M}_i$ and $\mathcal{N}_i$ in circuit $\mathcal{C}_i$. The question is, what does "similar" mean here. Our interpretation 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 which we can use. 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. Without this I/O type distribution, any pair of terminals can be hooked up, leaving too much freedom to deal with.

The above is a brief description of these statistics and reasons why they are important to us. Based on these three statistics, we can write the psudo-code for finding missing cells, set $\mathcal{M}_a$ as following:

```
1. AddCell(circuit C_i, p_c) {
2.     Get these three statistics discussed above.
3.     n_c = p_c · |M_i|
4.     for i = 1 to n_c
5.        Create a cell data structure.
6.        Determine the size of this cell.
7.        Determine number of terminals on this cell
8.        Determine each terminal's type.
9. }
```

In Step 6, 7 and 8, we make use of those three statistics created in step 2.

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, Similarity for graphs is very hard to define.

In the actual implementation, we use only one statistic for the set of nets, the number of terminals in each net. It is not clear what other statistics are also good, so we stick to this one for now. Even after deciding to use only one statistic, it is still difficult 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 3, 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. 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. There is nothing we can do in this situation unless we add some new cells. In this case, we will just stop adding new nets.

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*.

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 interconnected and rarely is there a connection between new cells and old cells.

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*.

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.

4 Experimental Results and Explanation

Fusion and *abstraction* 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 1 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 1, circuit Nprimary1 is the MCNC benchmark circuit primary1 without I/O pads, and Nprimary2 and Nstruct are benchmark primary2 and struct without I/O pads, respectively.

We feed the reconstructed circuit to the high quality placement tool called NRG [7]. Since this is a statistical approach, we needed to run a good number of trials to get the data. So we run NRG in fast mode.

Circuit	# Cells	# Pins	# Nets	WL
Nprimary1	750	3062	1156	642853
Nprimary2	2907	11681	3671	3215512
Nstruct	1888	5407	1920	374940

Table 1: Statistics for tested benchmark circuits.

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 3). 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 2 shows the total wirelength results achieved from the reconstructed circuit $Nprimary1'$. For each degree of incompleteness, column "WL" shows the average total wirelength over all trials; column "dev." shows the standard deviation of all trials, and "%err" shows the percent error compare to the real value from the original complete 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. 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 $\mathcal{C}_i$, the increase of incompleteness really does not affect the degree of randomness in reconstructing circuit $\mathcal{C}'$. That is why the standard deviation does not increase along with the percentage of incompleteness.

Similar results for Nprimary2 and Nstruct were obtained from the experiments. Table 3 is a summary table of these three circuits. All the percent errors in Table 3 are the average percent errors among these three circuits.

From experimental results shown above, we can conclude that *abstraction*'s prediction is much more accurate than *fusion*'s. That is very surprising and counter-intuitive.

The reason why *abstraction* is better than *fusion* is because *fusion* mantains the original properties of the circuit.

A big conclusion from this is that the netlist topology is very important. It will have a huge effect on the total wirelength. In this particular *placement with incomplete data* problem, we should not try to change the circuit topology of existing netlist.

% in	Fusion			Abstraction		
	WL	dev.	%err	WL	dev.	%err
0%	642853	6782	0%	642853	6782	0%
5%	740452	11697	15%	609263	9470	-5%
10%	874490	14284	36%	605948	6398	-6%
15%	941024	12486	43%	593153	7697	-8%
20%	1080158	16305	68%	597267	4250	-7%
25%	1202426	10309	87%	599750	8384	-7%
30%	1231592	11368	92%	560518	9623	-13%

Table 2: Wirelength results for Nprimary1 after reconstructing the circuit.

% of incomp.	%err for fusion	%err 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%

Table 3: Average wirelength prediction results for all three testing circuits.

5 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* as described in detail in Section 2. We want to see how good they can predict the final wirelength/ chip area using only an incomplete netlist and library.

Experimental results in Section 4 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.

References

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

[2] M. A. Breuer. "A Class of Min-cut Placement Algorithms". In *Design Automation Conference*, pages 284–290. IEEE/ACM, 1977.

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

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

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

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

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

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

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