Performance Driven Placement with Global Routing For Macro Cells

Andrew Lim† Yeow Meng Chee† Ching-Ting Wu*

† Information Technology Institute, 71 Science Park Drive, Singapore 0511
* 3M, St. Paul, Minnesota 55109

Abstract

In this paper, we present an effective performance driven placement with global routing algorithm for macro cells. Our algorithm is a hierarchical, divide and conquer, quad-partitioning approach. The quad-partitioning routine uses the *Tabu Search* technique. Our algorithm uses the concept of proximity of regions to approximate the interconnection delays during the placement process. In addition, our algorithm can handle modules whose positions are fixed or are restricted to a particular subregion on the layout frame. Our experimental results indicate the superiority of our placement in terms of quality of solutions and run times when compared to Lin [9].

1 Introduction

The advances in VLSI technology resulted in circuit devices with smaller feature sizes, switching delays and driving strength. The decrease in driving strength increases the propagation delay of signals through the wires connected to the devices. Consequently, interconnection delay becomes a major contributing factor [1, 11] to circuit performance. It was reported in [1, 11] that interconnection delays can be more than 50% of the circuit delay in some circuits. Hence, interconnection delays can no longer be ignored, and performance driven placement problem has become an important research topic in CAD for VLSI [9, 1, 11, 2, 10, 7, 8, 12].

Due to the immense complexity of the VLSI layout problem, it is normally divided into subproblems like partitioning, floorplanning, placement, global routing, detailed routing and layout compaction. Unfortunately, all these subproblems are still NP-hard.

Placement algorithms that incorporate interconnection delays information into consideration are known as performance driven placement algorithms. Placement is often considered to be the most important stage in performance driven layout as it determines the geometric locations of modules, thus, indirectly determining lower bounds of the interconnections. However, placement algorithms that do not consider global routing will over-estimate the performance of the circuit as numerous detours of interconnections usually take place due to congestion. Worst of all, it is possible that such a placement cannot be realized at all as prior planning for routing is not done during placement.

In this paper, we consider the placement with global routing problem. The placement problem is to determine the positions and orientations of the modules on a plane based on information of the modules and their interconnections. The global (also known as loose) routing problem is to decide approximately which regions the interconnections should be routed. The objective of our algorithm is to obtain a placement of modules of a circuit together with the consideration of global routing so that we can route the circuit with the predicted good performance during detailed routing. Our algorithm is a hierarchical, divide and conquer, quad-partitioning approach. The quad-partitioning is done using the *Tabu Search* meta-heuristic. We use the concept of proximity of regions to approximate interconnection delays during the placement process. Our delay approximation will progressively be more accurate as the regions that are being examined become smaller. In addition, our algorithm can handle modules whose locations are fixed, or are restricted to a particular subregion in the layout frame. Such a flexibility will allow the algorithm to handle placements with a mixture of standard cells and macro cells.

0-8186-2610-0/92 $3.00 © 1992 IEEE

2 Concept of Tabu Search

Like simulated annealing, *tabu search* is a general combinatorial optimization technique. It is proposed by Glover [3, 5, 4, 6] and has found applications in many areas like graph theory, mixed integer programming problems, scheduling, networks etc.

Tabu search is essentially a "meta-heuristic" superimposed on another heuristic. The higher level heuristic organizes and directs the operations of the subordinate one. Although tabu search and simulated annealing share the same property of being a general iterative improvement technique for finding good solutions to combinatorial optimization problems, the former does not resort to pure randomization to conquer intractability nor does it take the conservative approach that a proper rate of descend (i.e. cooling) will lead us to a good local optimum, hopefully close to the global one. Instead, it takes a more aggressive approach. Tabu search proceeds on the assumption that there is no value in choosing an inferior solution unless it is absolutely necessary to do so, as in the case of getting out of a local optimum. At each iteration of the search, it selects the best neighborhood solution. This is unlike hill climbing as it accepts the best neighborhood solution even if it gives a worse solution. Thus the algorithm never runs out of choices for the next move. However, this approach may introduce cycling in the algorithm, thus trapping the algorithm at locally optimal solutions. So structures known *tabu lists* and *aspiration function* are introduced. These two structures keep information about past moves in order to constrain and diversify the search for good solutions. Figure 1 provides the general framework of tabu search.

Input
The problem to be solved
Definitions
X : Set of feasible solutions
f : Objective Function
$N(x)$: Neighborhood of $x \in X$
T : Tabu List(s)
A : Aspiration Function
max : Max # of iterations between improvement
Initialization
Set $i = 0$
Generate an initial solution $x_i \in X$
Initialize tabu list(s) T
Initialize aspiration function $A()$
Set $best = x_i$, $bestcost = f(best)$ and $besti = i$
Body
while $i - besti < max$ do
begin
$i = i + 1$
locate the best x_i in $N(x_{i-1})$
where x_i does not satisfy tabu conditions or
if it satisfies tabu conditions but aspiration function
overrules the tabu conditions
if $f(x_i) < bestcost$ then
begin
$best = x_i$
$bestcost = f(best)$
$besti = i$
end
update tabu list(s) T
update aspiration function A
end
Output
best and *bestcost*

Figure 1: General framework of Tabu Search

3 Terminology and Problem Formulation

Figure 2 clarifies some definitions and terms we shall use below. The layout frame is rectangular in shape. It is the region where modules, except in the case of I/O pins, are placed. Surrounding the layout frame are the I/O pads. Only I/O pins can be assigned to I/O pads. Modules are rectangular in shape and associated with each of them is a delay value called module delay.

3.1 Circuit Topology and Modeling

Net list is a list containing the interconnections between modules. We assume that the modules and in-

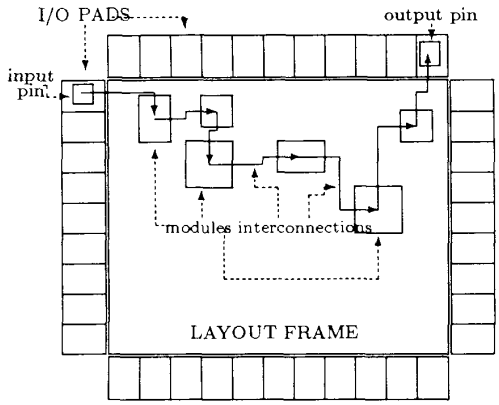

Figure 2: Layout

terconnections can be modeled by a directed acyclic graph (DAG) $G=(V,E)$. We associate a node to each module. If a module is connected to another, there will be a directed edge between them depending on the direction of the signal flow. So, all input pins have in-degree of zero and output pins have out-degree of zero. Other modules, which are not I/O pins, have non-zero in-degree and out-degree nodes. A path will always start and end with I/O pins. This can easily be extended to the situation where there are synchronizing elements, like latches or flip-flops, in the circuit. In that situation, each synchronizing element is split into 2 modules such that all the incoming connections of the synchronizing element will be the incoming connections of one module with no outgoing connection, and the outgoing connections with be the outgoing connections of the other module with no incoming connection. With the modification, a path begins with an input pin or an output pin of a synchronizing element and ends with an output pin or an input to a synchronizing element. Path delays can then be computed accordingly.

3.2 Delay Model

A path starts with an input pin and ends with an output pin. The modules in Figure 2 form a path. Path delay is the sum of all module delays and interconnection delays on the path.

The delay of a path $P=< M_{i_1} \rightarrow M_{i_2} \rightarrow \ldots \rightarrow M_{i_k} >$ is estimated as:

$$Delay(P) = \sum_{1 \leq j \leq k} MD(M_{i_j}) + \sum_{1 \leq j < k} ID(M_{i_j}, M_{i_{j+1}})$$

where $MD()$ is the module delay (given for each module) and $ID()$ is the interconnection delay. Note that M_{i_1} and M_{i_k} are I/O pins.

Interconnection delay ($ID()$) is:

$$ID(M_i, M_j) = K_{RC} \times Dist(M_i, M_j)^2 \qquad (1)$$

where $Dist(M_i, M_j)$ is the Manhattan distance between module M_i and module M_j. K_{RC} is a constant determined by the average resistance and capacitance per unit length of the interconnection over all routing layers.

3.3 Region Constraints

Some modules may be placed only in certain regions. Constraints on the possible locations of modules is to handle the following situations :

- Movable I/O pins. Some I/O pins' positions maybe fixed, others may be assigned to any of the available I/O pads.
- Locations of some modules maybe already fixed. In some cases, where the layout consists of macro cell modules and standard cell modules or digital and analog modules, the regions in which they are supposed to be placed may have already been decided. So, such a flexibility will enable us to perform placements in such situations.

3.4 Formulation for Performance Driven Placement with Global Routing

In our performance driven placement with global routing problem, we are given:

- a set of modules with fixed rectangular geometries and delays (modules may be I/O pins).
- a layout frame where the modules are to be placed.
- constraints on the possible positions of some modules (if any).
- a net list specifying the interconnections between modules.
- resistance and capacitance attributes of the interconnections.

The objective of our algorithm is to determine the positions of modules in and on the layout frame such that

Step 1: Put the entire placement region into queue Q.

Step 2: If Q is empty goto Step 7, else do

- Get region R from Q.
- Divide R into 4 subregions R_1, R_2, R_3, R_4.

Step 3: Generate an initial solution.

Step 4: Calculate path delays.

Step 5: Perform Quad-Partitioning.

Step 6: If R_i, $i = 1, 2, 3, 4$, has k or more modules in it, add R_i into Q, else add to branch and bound queue, BBQ. Goto Step 2.

Step 7: For each region, R, in BBQ perform a branch and bound search for the best solution for that region.

Step 8: Shift and adjust to remove overlaps.

Figure 3: Outline of our algorithm

1. global routing is performed.
2. the delay of the longest path is minimized.
3. there is no overlap between modules.

4 The Algorithm

The outline of our algorithm is given in Figure 3. Details of the individual steps are given in the subsections below.

4.1 Division

Given a region R, let the 2 corners of the region be (X_{min}, Y_{min}) and (X_{max}, Y_{max}). We divide the region R into 4 subregions R_1, R_2, R_3, R_4 by cutting R horizontally and vertically through its center point, $(\frac{X_{min}+X_{max}}{2}, \frac{Y_{min}+Y_{max}}{2})$ (see Figure 4).

4.2 Initial Solution

The initial solution is obtained by randomly assigning modules with region constraints first into the four subregions, and then followed by all other modules. The assignment is done in such a way so that the sum of areas of modules in each subregion is almost equal.

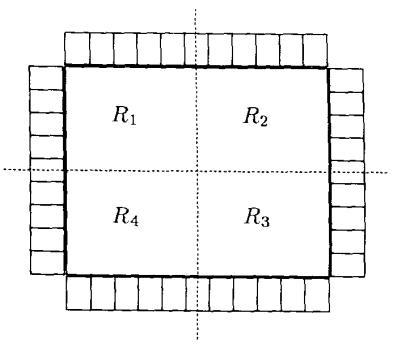

Figure 4: Division of a region

4.3 Path Delay Estimation

First, we need to estimate the length of all interconnections. We define X_c^R to be the center of region R. If the edge connects 2 modules in the same region, then its length is estimated as :

$$\frac{(X_{max}^R - X_{min}^R) + (Y_{max}^R - Y_{min}^R)}{3.5}$$

If the modules are in different regions, R_1 and R_2, then the length is :

$$|X_c^{R_1} - X_c^{R_2}| + |Y_c^{R_1} - Y_c^{R_2}|$$

After the length of each edge is computed, its delay can be estimated using equation (1) in Section 3.2. Finally, for each edge e, we compute :

1. longest path length passing through it, L_e^{max}
2. number of paths through it, N_e.
3. average lengths of all incoming paths and outgoing paths.

Note that this step can be done by a forward and backward topological sweep. It takes no more then $O(|E| + |V|)$ time, where $G=(V, E)$ is the directed acyclic graph modeling the circuit topology (see Section 3.1).

4.4 Quad-Partitioning Using Tabu Search

This the most important part of the algorithm. The quality of the final solution will depend on the quality of solution obtained at each level of the partitioning. Quad-partitioning is chosen instead of doing 2

bi-partitioning because the placement problem is a 2-dimensional problem and bi-partitioning is essentially one dimensional.

Our partitioning algorithm is based on the *tabu search* approach. We define the cost function to be:

$$\alpha F_A + \beta F_B + \gamma F_C + \delta F_D + \epsilon F_E$$

where $\alpha, \beta, \gamma, \delta$ and ϵ are constants.

F_A measures the imbalance of area usage of the 4 subregions, R_1, R_2, R_3 and R_4 created. It is given by the formula :

$$\frac{\max_{i=1}^{4}\{\sum_{m \in R_i} Area(m)\} - \min_{i=1}^{4}\{\sum_{m \in R_i} Area(m)\}}{Area(R_1) + Area(R_2) + Area(R_3) + Area(R_4)}$$

F_B is the total number of edges that pass through all the four boundaries. An edge that connects modules in R_1 and R_3 or R_2 and R_4 will contribute to $\frac{1}{2}$ an edge to all the 4 boundaries.

F_C measures the imbalance of edges through the boundaries. Formulation is analogous to F_A. It is given as the maximum difference between the number of edges across any of the four boundaries.

F_D is the square of the summation of L_e^{max} where e is an edge that spans across one of the four boundaries. If an edge connects modules in R_1 and R_3 or R_2 and R_4, then L^{max} of the edge is added twice to the total.

F_E measures the criticality of the edges the spans across one of the four boundaries. By criticality, we mean how many long paths pass through this edge and what is their average length.

We use the general framework of the tabu search technique as outlined in Figure 1. The neighborhood configurations of the current configuration are those configurations that are reachable from the current configuration by the two types of moves given below:

- Move one module from one subregion to another.
- Exchange two modules in different subregions.

The neighborhood configurations of the current configuration is obtained by performing one of the above moves. Among all possible moves, we choose the best one that does not satisfy the tabu conditions or if the aspiration function overwrites the tabu conditions. There can be at most $O(n^2)$ moves at each level of the subdivision.

A move will only affect the incoming and outgoing edge lengths of the module/modules being moved. Note that recomputing everything is not necessary to determine the cost of the new configuration. All that is needed is to recompute those edge lengths and path lengths affected by the move. In practice, this takes O(1) time (in the worst case $O(|V| + |E|)$ time is required).

The *tabu list* stores the last 10 modules being moved. The length of the tabu list is chosen to be 10 as that seems to give consistently good partitions with the least time required to check for tabu status, although any length between 10 to 15 will suffice. The *aspiration function* stores the cost of the configuration when the module was last in that subregion. The aspiration function will overrule the tabu condition if the current move decreases the aspiration function value for the module in that subregion.

4.5 Branch and Bound Search

We fixed the constant k to be 10. When the number of modules in any region is less than 10, we perform a branch and bound search for the best possible arrangement of the modules in that region.

4.6 Shifting and Adjustment

Minor overlaps may happen after the first seven steps. We will adjust the placement without changing the relative positions of the modules to eliminate the overlaps. This is done by starting at a corner of the layout frame and considering module packing along the direction of the diagonal from that corner. After all overlaps are removed, minor adjustments are made to further improve the layout.

5 Experimental Results

We implemented a preliminary version of our algorithm in C and ran our experiments on a SUN SPARCstation. We use the test circuits in Lin [9]. Table 1 provides the characteristics of the test data. Table 2 tabulates our results. Our algorithm is faster then the algorithm proposed in [9]. This is obvious as in [9], the performance of their algorithm depends on the number of paths in the circuit, and the number of paths can grow exponentially in $|V|+|E|$. The length of our longest paths are only slightly superior than those in [9]. This is because in our algorithm, global routing is already taken care of. In Lin's placement, the only objective is to minimize the longest path length without considering if the routing can actually be realized. So, if global routing is performed on their layout, the solution could be much worse.

	# of modules	# of nets	module density	# of paths
ckt 1	10	26	0.3	18
ckt 2	15	32	0.3	23
ckt 3	20	43	0.4	51
ckt 4	20	53	0.4	81
ckt 5	30	75	0.4	305
ckt 6	40	104	0.5	1314
ckt 7	40	104	0.6	1314
ckt 8	40	104	0.7	1314
ckt 9	60	153	0.75	1274
ckt 10	100	201	0.5	7438

Table 1: Characteristics of test cases

		Lin's Algorithm		Our Algorithm	
	lower bound	circuit delay	run time	circuit delay	run time
ckt 1	239.66	326.73	2.12	320.6	1.01
ckt 2	296.02	345.33	11.45	349.2	3.21
ckt 3	424.62	594.06	27.79	593.8	7.66
ckt 4	424.62	611.62	31.70	615.1	8.89
ckt 5	1241.47	1581.8	161.09	1500	41.6
ckt 6	1266.96	1570.8	841.25	1550	215
ckt 7	1266.96	1703.3	849.19	1679	256
ckt 8	1266.96	1684.5	846.65	1654	289
ckt 9	1507.67	2216.9	1637.4	2210	643
ckt 10	2469.03	3012.6	16832	3000	2412

Table 2: Comparisons with Lin's algorithm

6 Conclusion

In this paper, we proposed an effective performance driven placement algorithm that takes global routing into consideration. Our experimental results indicate its usefulness. Our algorithm uses a hierarchical, divide and conquer, quad-partitioning concept, incorporating the global routing and performance issues into the quad-partitioning routine. Our algorithm also uses a novel approximation scheme to estimate net delays. Our approximation becomes progressively more accurate as the regions become smaller. In addition, our algorithm allows region restrictions for modules to handle fixed and movable I/O pins and possibly analog and digital placement if the analog and digital regions of the layout frame are already pre-defined. In the quad-partitioning routine, we applied the concept of *Tabu Search* to obtain good results. The results in this paper indicate the potential of applying the tabu search technique to other VLSI CAD problems.

7 Future Work

We are extending this work to handle pin to pin delay computations. In addition, we hope to use the tabu search framework to develop partitioning, floorplanning, detailed routing and layout compaction subsystems and integrate these subsystems into a complete layout tool.

References

[1] W. Donath and et.al. Timing Driven Placement Using Complete Path Delay. In *27th Design Automation Conference*, pages 84–89, 1990.

[2] A.E. Dunlop, V.D. Agrawal, and et al. Chip Layout Optimization Using Critical Path Weighting. In *21st Design Automation Conference*, pages 133–136, 1984.

[3] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comput. and Ops. Res*, 13(5):533–549, 1986.

[4] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.

[5] F. Glover and H.J. Greenberg. New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European Journal of Operational Research*, 39:119–130, 1989.

[6] Fred Glover. Tabu Search II. Technical Report CAAI-TR-89-05, University of Colorado at Boulder, 1986. Center For Applied Artificial Intelligence.

[7] M.A.B. Jackson and E.S. Kuh. Performance-Driven Placement of Cell Based IC's. In *26th Design Automation Conference*, pages 370–375, 1989.

[8] M.A.B. Jackson, A. Srinivasan, and E.S. Kuh. A Fast Algorithm for Performance-Driven Placement. In *International Conference on Computer-Aided Design*, pages 328–331, 1990.

[9] I. Lin and David Du. Performance-Driven Constructive Placement. In *27th Design Automation Conference*, pages 103–106, 1990.

[10] R.Nair P.S. Hauge and E.J Yoffa. Circuit Placement for Predictable Performance. In *ICCAD-87*, pages 88–91, 1987.

[11] S. Sutanthavibul and E. Shragowitz. An Adaptive Timing-Driven Layout for High Speed VLSI. In *27th Design Automation Conference*, pages 90–95, 1990.

[12] C.T. Wu, Andrew Lim, and David Du. An Effective Timing-Driven Placement Algorithm For Macro Cells. In *5th International Conference in VLSI Designs*, 1992.