Timing Driven Placement for Large Standard Cell Circuits

William Swartz
TimberWolf Systems, Inc.
10880 Cassandra Way
Dallas, TX 75228

Carl Sechen
Department of Electrical Engineering
University of Washington
Seattle, Washington 98195

1. Abstract

We present an algorithm for accurately controlling delays during the placement of large standard cell integrated circuits. Previous approaches to timing driven placement could not handle circuits containing 20,000 or more cells and yielded placement qualities which were well short of the state of the art. Our timing optimization algorithm has been added to the placement algorithm which has yielded the best results ever reported on the full set of MCNC benchmark circuits, including a circuit containing more than 100,000 cells. A novel pin-pair algorithm controls the delay without the need for user path specification. The timing algorithm is generally applicable to hierarchical, iterative placement methods. Using this algorithm, we present results for the only MCNC standard cell benchmark circuits (*fract*, *struct*, and *avq.small)* for which timing information is available. We decreased the delay of the longest path of circuit *fract* by 36% at an area cost of only 2.5%. For circuit *struct*, the delay of the longest path was decreased by 50% at an area cost of 6%. Finally, for the large (22,000 cell) circuit *avq.small*, the longest path delay was decreased by 28% at an area cost of 6% yet only doubling the execution time. This is the first report of timing driven placement results for any MCNC benchmark circuit.

2. Introduction

A placement algorithm must control the wire lengths on a set of *critical signal paths*. If the parasitic delays are large enough for these critical paths, the circuit may not function properly. In general, there is an upper bound T_{ub} on the time t_r that a signal may propagate from the input to the output for which the circuit still functions,

$$t_r \leq T_{ub} \tag{1}$$

The delay from input to output can be decomposed into two pieces: component or gate delay and parasitic delay. The placement algorithm must ensure that all wiring parasitics are within their bounds.

Figure 1 shows an example of a digital logic circuit. There are three inputs (A, B, C) and two outputs (D, E).

The gate and wiring delays for this circuit may be represented using a timing graph as shown in Figure 2. The timing graph is a weighted directed acyclic graph (*dag*). The nodes of the graph represent the signal pins of the circuit. The edges of the graph connect the pins. Each edge has a weight corresponding to the propagation delay. There are two types of edges: internal and external. The internal edges are signal paths which exist between pins of a gate. The weights of these edges are fixed by technology and device physics. The external edges denote paths created by the signal network. The weight of these edges depends on the length of the interconnect. The goal of the placement algorithm is to satisfy the time bounds for all signal paths through the circuit.

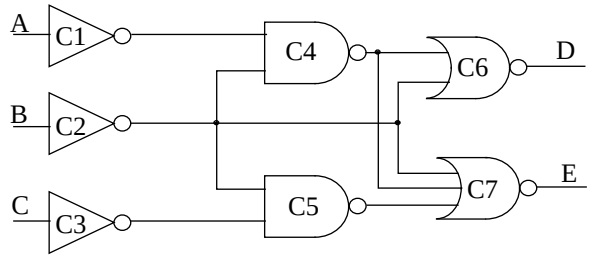

Figure 1. Example circuit.

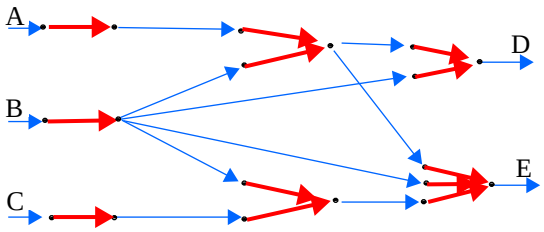

Figure 2. Timing graph for the example circuit of Figure 1. The nodes of the graph are the signal pins. The edges of the graph connect the pins. There are two types of edges. The thick edges are signal paths through the gates whereas the thin lines denote the signal nets.

Often the designer knows the timing constraints between a primary input pin and a primary output pin (such as B and E in Figure 2) but does not know the gates or nets of the paths between them. In Figure 2, there are three unique paths between pins B and E. The longest time among the three paths will set the upper bound on the time delay. However, some of these paths may be logically or temporally incompatible and are therefore, *false paths*. A timing constraint for a false path is meaningless since the path can never be switched or *sensitized*. A false path unnecessarily constrains the placement problem.

Previous timing driven placement schemes have either been net-based [1] [4] [7] [9] [12] [14] [19] [20] or path-based algorithms [2] [5] [8] [11] [14] [15] [17] [18]. Net-based algorithms seek to control the delay on a signal path by putting a separate constraint on each net in the signal path. This usually severely overconstrains the placement algorithm since some of the nets in the path may be well shorter than their bounds implying that the other nets could accommodate more delay than their bounds without having excessive delay on the path. Consequently, the placement quality suffers badly since the algorithm seeks to make nets shorter than they really need to

32nd ACM/IEEE Design Automation Conference ®

be. Path-based approaches correctly model the problem but have previously been limited to small problems. As a consequence, path-based approaches have not been adopted in industry.

In this paper, we present a path-based timing driven placement algorithm which can handle very large integrated circuits. The paper is organized as follows. In Section 3, we review the necessary fundamentals of the placement algorithm upon which we base our new approach to timing driven placement. The new timing optimization algorithm is presented in Section 4. In Section 5, we describe how the timing optimization algorithm was extended for hierarchical placement. Finally, in Section 6, we present our results.

3. Review of the Basic Placement Algorithm

We felt that we could develop the most effective timing driven placement algorithm by adding the new timing optimization algorithm to the most effective row-based placement algorithm available, namely TimberWolfSC version 1. In this section we review the relevant aspects of the TimberWolfSC version 1 placement algorithm.

Figure 3 shows the hierarchical placement methodology described in [16], which combined a new clustering technique with a new approach to simulated annealing. The original netlist is hierarchically clustered into various levels of netlists. Then the new approach to simulated annealing was used to place those various levels of netlists.

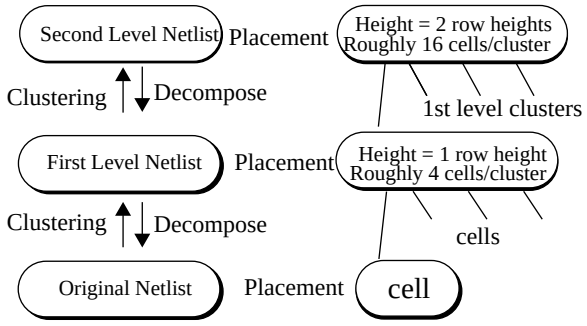

Figure 3. Hierarchical Placement

In the clustering stages, the original netlist is condensed into the first and then the second level netlists. The produced clusters in the higher level netlists have similar size, which greatly aids the annealing placement stages. In the placement stages, the condensed second level netlist is placed using simulated annealing at the higher temperatures. Then the second level netlist is decomposed back to the first level netlist. Cells of the lower level netlist are randomly placed within the range of the cluster to which they belong in the higher level netlist. At the new lower level, these cells may then move outside the bounds of the higher level cluster. The first level netlist is then placed at the middle temperatures. Next, the first level netlist is decomposed back to the original netlist, and the original netlist is placed at the lower temperatures. There are two clustering stages and three placement stages. The right hand side of Figure 3 shows some typical values.

4. Timing Optimization Algorithm

We now present our new timing optimization algorithm. We will first describe how the delay on a critical path is computed. We then show how critical path delays enter into the computation of the simulated annealing cost function. A description of the various ways in which timing constraints may be specified by users and how the algorithm efficiently handles the potentially huge numbers of critical paths. Finally, we show how we handle sequential circuits.

4.1 Computing the Delay on a Critical Path

The arrival time (delay) T_a for a path p is the summation of all the net delays D_n for the path.

$$T_a(p) = \sum_{\forall n \in p} D_n \tag{2}$$

The delay for a single net n is the sum of the intrinsic gate delay T_n associated with the driver of the net n, and the product of the equivalent driver resistance R_n, and the total load capacitance seen by the driver C_n.

$$D_n = T_n + R_n C_n \tag{3}$$

The total capacitance for a net has two components: gate input capacitance C_{G_n} and parasitic capacitance C_{p_n}.

$$C_n = C_{G_n} + C_{p_n} \tag{4}$$

During placement, we can estimate the parasitic capacitance using the half-perimeter bounding-box metric where C_{L_x} and C_{L_y} are the capacitance per length in the x and y directions respectively, and $S_x(n)$ and $S_y(n)$ are the horizontal and vertical spans, respectively, of the bounding box of net n.

$$C_{p_n} = C_{L_x} S_x(n) + C_{L_y} S_y(n) \tag{5}$$

Substituting (4) and (5) into (3), we get:

$$D_n = T_n + R_n C_{G_n} + R_n [C_{L_x} S_x(n) + C_{L_y} S_y(n)] \tag{6}$$

We can precompute the terms in the summation which do not depend on wire length by defining:

$$K_p = \sum_{\forall n \in p} [T_n + R_n C_{G_n}] \tag{7}$$

This results in the following simplified equation for the arrival time:

$$T_a(p) = \sum_{\forall n \in p} R_n [C_{L_x} S_x(n) + C_{L_y} S_y(n)] + K_p \tag{8}$$

4.2 Cost Function

We now show how critical path delays enter into the computation of the simulated annealing cost function. The penalty due to timing for a path p is zero unless $T_a(p)$ is greater than $T_{ru}(p)$ in which case

$$P(p) = T_a(p) - T_{ru}(p) \tag{9}$$

where the user has specified an upper bound (T_{ru}) on the required arrival times.

The total timing penalty P_T is just the sum over all critical paths:

$$P_T = \sum_{p=1}^{N_p} P(p) \tag{10}$$

The simulated annealing cost function C consists of two terms. The first term is the total wire length, represented by W. The second term is the timing path penalty function P_T.

$$C = W + \lambda P_T \tag{11}$$

A very large value of λ will surely satisfy all timing constraints (if they can be satisfied) but will result in poor values of W. Conversely, too small a value of λ will yield very good values of W but with some of the time constraints remaining unsatisfied. We therefore experimented to find a relationship for λ which did not degrade W, and correspondingly the chip area, but which satisfies all the feasible path constraints. Since the ratio of paths specified to the total number of nets will vary from zero to a large number from circuit to circuit and run to run, we had to ensure that the timing penalty term was *felt*, regardless of its absolute value. The best results were obtained when we assigned

$$\lambda = 3 \cdot \frac{\overline{\Delta W}}{\overline{\Delta P_T}} \tag{12}$$

where $\overline{\Delta W}$ is the average change in wire length and $\overline{\Delta P_T}$ is the average change in the timing penalty measured during the first iteration (line 5 in Figure 4) of the simulated annealing algorithm. This implies that the changes in the timing penalty are (in some sense) three times as important as the changes in the wire length.

It should be noted that there is no fundamental reason for using the lumped capacitance model in the timing analysis. Other models which incorporate resistance of the routing layers can be used as well.

4.3 Specification of Critical Paths

We now describe the ways in which timing constraints may be specified by users and how the algorithm efficiently handles the potentially huge numbers of critical paths. The simplest way for users to specify critical paths is by listing the sequence of nets comprising a path. However, often users do not know which paths are critical through the circuit. Instead, they are concerned with the timing delays between the primary inputs and primary outputs of the integrated circuit. In this latter case, our timing optimization algorithm must find all the relevant critical paths.

The number of critical paths that could potentially violate delay specifications can be truly large and unwieldy. Not only might the user list a vast number of critical paths, but between a single pair of I/O pins there may be huge number of non-false signal paths. If all such paths were entered into the cost function in the manner described in the previous subsection, the computation time for a placement run can go up by orders of magnitude compared to the unconstrained case. The secret to controlling the run time hit is to have the annealing algorithm pay attention to only the K costliest paths (those paths yielding the largest penalty as in (9) in the previous section) at any given time. Of course, it is vitally important that this list of K costliest paths be updated often enough during the course of an annealing run. And updating the list more often than necessary wastes computation time.

We modified the simulated annealing algorithm as shown in Figure 4. The first step of the algorithm reads false paths designated by the user. The user may remove any false paths by enumerating them in a file. This is similar to the TA timing analyzer which allowed the user to add *delay modifiers* to indicate that a path was not possible [10]. The false paths may be obtained by using external timing analyzers.

In the second step, a timing graph for each specified pin-pair is constructed. The timing graph construction routine creates a directed acyclic graph (dag). It is important that the graph be a dag; the dag's special properties are exploited in the M shortest path algorithms. It allows the use of the PERT

Algorithm Simulated-Annealing-With-Timing(M,K)

```
1   {FP} ← readFalsePaths()
2   buildTimingGraph (V, E)
3   determine initial temperature
4   start with a random initial placement of the cells
5   for each of the 150 iterations of the outer loop do
6     do
7       generate a move
8       determine whether to accept/reject based on Δ C
9       periodically adjust the temperature
10    until iteration is complete
11    for each d = (s, t) ∈ P do
12      {T_a}_d ← findMLongestPaths (s, t, M)
13    discard all but the K costliest paths
14  until cost cannot be reduced any further
```

Figure 4. Modified simulated annealing algorithm with timing analysis. After each iteration of the outer loop, a new set of timing constraints is generated. An upper bound of M such constraints are generated for each pin-pair. The key new steps in the timing driven placement algorithm are enclosed by the boxes.

algorithm to calculate the longest path in the graph in $O(V+E)$ time. It further allows the use of Dreyfus's method to compute all the remaining M longest paths of a single pin-pair in time complexity $O(Mn\log n)$ [3][13]. This is in contrast to Yen's method for general networks with time complexity $O(Mn^3)$ [21]. Dreyfus's method finds paths in which repeated nodes are admissible paths in a general network. We seek paths without repeated nodes. Since the graph is a dag, repeated nodes are not possible.

After each iteration of the outer loop, a new set of paths to be monitored is created in lines 11 through 13. For each specified pin-pair, the M longest paths of this pair are found by negating the edge weights of the timing graph and running the M shortest paths algorithm. This is possible since the timing graph is a dag. These paths are now treated as time critical paths and (9) is used to determine the cost. In line 13, all but the costliest K time critical paths are discarded. The parameter K controls the number of paths to be monitored during a single execution of the outer loop. We have found empirically that updating the list of K costliest paths every iteration is sufficient to meet the timing specifications. Updating it more often only increases the total CPU time but does not yield better timing performance. Updating it appreciably less often can cause the timing penalty function to fail to converge for some circuits.

4.4 Sequential Circuits

Until now we have described how to optimize a strictly combinational circuit. In these circuits, the input/output pin-pairs are relatively easy to discover. We simply need to examine the cross product of primary inputs and primary outputs. The cross product yields $O(mn)$ pin-pairs, assuming m inputs and n outputs. Many of these pin-pairs have no path between them. To save computation time, we preprocess the pin-pairs before the placement process begins and retain only those pairs of I/O pins which have at least one path between them, thereby avoiding unneeded searching in the timing graph. We employ a similar technique for synchronous circuits.

In a sequential circuit, the cycle time is determined by the longest delay on a combinational path between an output of

one storage element and the input of another (or the same) storage element. In timing driven placement we seek to minimize (or control) the longest path delay between storage elements. To discover the connected pairs of storage elements, we examine each of the outputs of a storage element in turn. Each output is used as a source in the PERT algorithm; all inputs of storage elements are used as targets. The longest path algorithm is executed. If a target has been labeled by the algorithm, a path exists between these pairs. We output this pair and continue until all storage element outputs have been exhausted. We were able to process the 4007 sources and 4277 targets found in the benchmark circuit *avq.small* in less than 800 seconds (a small portion of the overall run time) on a DECstation 3000/400. The timing graph for this circuit had 64074 nodes and 80508 edges. The preprocessing program located the 33,829 viable pin pairs out of a possible 17 million in this circuit.

5. Hierarchical Placement

Recall from Section 3 and Figure 3 that our placement algorithm is hierarchical. That is, during the first two of the three placement stages, clustered (or condensed) versions of the original flat netlist are used in the simulated annealing algorithm.

If the only critical paths are those actually listed by the user, it is straight forward to incorporate timing in the first two (clustered) placement stages. The lengths of nets which are fully internal to a cluster are unknown and rather short, and hence the delays due to these nets can be successfully approximated to zero. Only the delays due to inter-cluster nets need to be computed by the timing optimization algorithm.

However, if pin-pair constraints are present, problems arise if one attempts to simplify the timing graph during clustering. The clustering will generally induce cycles in the simplified graph. Removal of the cycles will remove valid constraints.

For these reasons, we retain the flattened timing graph throughout all of the three placement stages if pin-pair constraints have been specified. Steps 11-13 of Figure 4 are modified slightly. As before, the delays due to intra-cluster nets are approximated to zero and hence their edge weights are set to zero in the flattened timing graph. All inter-cluster edges are updated with their parasitics based on the half-perimeter bounding box metric (as before). The graph is then searched for the longest M paths discarding all but the *K* costliest paths. These paths are then simplified by removing any intra-cluster edges. The simulated annealing based placement optimization proceeds for another iteration using this set of paths derived from the original timing graph.

6. Results

Three MCNC benchmark circuits contain timing information. The first two, *fract* and *struct*, have combinational paths between their primary inputs and primary outputs. The third, *avq.small* is a sequential circuit. Table 1 displays the characteristics of the benchmark circuits and their timing graphs

For circuit *fract*, the most interesting pair of I/O pins is *X* and *Z* with 17 unique paths between them. Table 2 compares the effect of varying *M*, the maximum number of longest paths generated for each pin pair, on the longest path in the circuit. In this case, none of the constructed paths were pruned (*i.e.* $K = \infty$). The results of using the pin-pair timing algorithm are dramatic; the longest path delay decreased by 36%. Among the 17 paths from *X* to *Z*, it was discovered that five paths were much longer than the others. Curiously, the best average results were obtained when we set $M = 5$. In this case, the longest timing path through the chip was reduced by 41%. Our experience suggests that this is because the algorithm was not distracted by the easily satisfiable paths.

Table 1 Circuit and timing graph characteristics.

Circuit	# cells	# nets	V[G]	E[G]
fract	149	171	486	588
struct	1952	1984	5535	7134
avq.small	21918	22178	64074	80508

A decrease in the delay for the other (lower delay) paths was also realized. The average delay in the circuit was reduced from 115 nanoseconds to 37 nanoseconds, a savings of 67%.

Table 2 compares wire length, area after global routing, and execution times for each case. When *M* is set to 1 and *K* is set to infinity, the longest path delay was reduced by 36%, with an area penalty of only 2.5% and an execution time increase of 16%. Generally, the area after global routing correlates with the wire length after placement. For this circuit, dynamically searching (at the beginning of each of the 150 iterations) and retaining only the longest among the set of paths for each pin pair yields excellent results with minimal increase in the execution time.

Table 2 Results for *circuit fract* ($K = \infty$).

	unconstrained	M=1	M=5	M=17
longest path	208ns	134ns	124ns	130ns
wire length	57276	60691	63073	61209
area (μm^2)	1.56×10^6	1.60×10^6	1.68×10^6	1.58×10^6
run time	670 sec	801 sec	966 sec	1201 sec

To further reduce the run time, the number of time critical paths retained at the start of each iteration was pruned to only the 5 costliest paths. Table 3 shows the delay of the longest path using pruning. Notice that the longest delay path did not increase (actually decreased). In Table 3, we see that the area after global routing also remained the same. In addition, the CPU time was reduced.

Table 3 Results for circuit *fract* using pruning.

	M=1, K=∞	M=1, K=5
longest path (ns)	134	131
wire length	60691	60322
area (μm^2)	1.60×10^6	1.60×10^6
run time (secs.)	801	750

In Table 4, the longest delay in the circuit *struct* was reduced from 907 nanoseconds to 449 nanoseconds, a reduction of 50%. The user needed no knowledge of the time criti-

cal paths in the circuit. The longest paths in the circuit were discovered and optimized automatically. Table 4 contrasts the wire length, chip area, run time, and memory utilization for the unconstrained and constrained cases. Here, a 6% increase in area results in a 50% decrease in longest path delay. Notice that the run time remains feasible and memory utilization is comparable.

Table 4 Results for benchmark circuit *struct.*

	no constraints	M=1, K = 2000
longest path (ns)	907	449
wire length	699144	734330
area (μm^2)	1.89×10^7	2.01×10^7
run time (secs.)	802	2175
memory	34 Mbyte	37 Mbyte

In Table 5, we present the results of the circuit *avq.small* using the hierarchical placement algorithm. In this case, a 6% increase in the area results in a 28% decrease in the longest path delay. Time and memory resources increased less than a factor of two over the unconstrained case.

Table 5 Results for benchmark circuit *avq.amall*.

	no constraints	M=1, K = 2000
longest path (ns)	1106	798
wire length	5.30×10^6	5.68×10^6
area (μm^2)	1.33×10^8	1.41×10^8
run time (secs.)	24049	47990
memory	43 Mbyte	61 Mbyte

7. Conclusions

We presented an algorithm for accurately controlling delays during the placement of large standard cell integrated circuits. Previous approaches to timing driven placement could not handle circuits containing 20,000 or more cells and yielded placement qualities which were well short of the state of the art. Our timing optimization algorithm has been added to the placement algorithm which has yielded the best results ever reported on the full set of MCNC benchmark circuits, including a circuit containing more than 100,000 cells. A novel pin-pair algorithm controls the delay without the need for user path specification. The timing algorithm is generally applicable to hierarchical, iterative placement methods. Using this algorithm, we present results for the only MCNC standard cell benchmark circuits (*fract*, *struct*, and *avq.small)* for which timing information is available. We decreased the delay of the longest path of circuit *fract* by 36% at an area cost of only 2.5%. For circuit *struct*, the delay of the longest path was decreased by 50% at an area cost of 6%. Finally, for the large (22,000 cell) circuit *avq.small*, the longest path delay was decreased by 28% at an area cost of 6% yet only doubling the execution time. This is the first report of timing driven placement results for any MCNC benchmark circuit.

References

[1] M. Burstein and M. N. Youseff, "Timing Influenced Layout Design," *Proc. 22th Design Automation Conference*, 1985, pp. 124-130.

[2] W. Donath, R. Norman, B. Agrawal, S. Bello, S. Han, J. Kurtzberg, P. Lowy, and R. McMillan, "Timing Driven Placement using Complete Path Delays," *Proc. Design Automation Conference*, June 1990, pp. 84-89.

[3] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Operations Research*, Vol. 17, 1969, pp. 395-412.

[4] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip Layout Optimization Using Critical Path Weighting," *Proc. 21st Design Automation Conference*, 1984, pp. 133-136.

[5] P. de Forcrand and H. Zimmermann, "Timing-Driven Auto-Placement," *Proc. Int. Conf on Comp. Design*, 1987, pp. 518-521.

[6] T. Gao, P. M. Vidya, and C. L. Liu, "A New Performance Driven Placement Algorithm," *Proc. Int. Conf. on Computed-Aided Design*, November 1991, pp. 44-47.

[7] T. Gao, P. M. Vidya, and C. L. Liu, "A Performance Driven Macro-Cell Placement Algorithm," *Proc. Design Automation Conference*, June 1992, pp. 147-152.

[8] T. Hasegawa, "A New Placement Algorithm Minimizing Path Delays," *Proc. Int. Conf. on Computed-Aided Design*, 1991, pp. 2052-2055.

[9] P. Hauge, R. Nair, and E. Yoffa, "Circuit Placement for Predictable Performance." *Proc. Int. Conf. on Computed-Aided Design*, 1987, pp. 88-91.

[10] R. B. Hitchcock, G. L. Smith, D. D. Cheng, "Timing analysis of computer hardware," *IBM Journal of Research and Development*, Vol 24, No. 1, Jan. 1983, pp. 100-105.

[11] M. Jackson and E. Kuh, "Performance-Driven Placment of Cell Based IC's," *Proc. Design Automation Conference*, June 1989, pp. 370-375.

[12] M. Jackson, E. Kuh, and M. Marek-Sadowska, "Timing-Driven Routing for Building Block Layout," *Proc. of Int. Symp. Circuits and Systems*, 1987, pp. 518-519.

[13] E. Lawler, Combinatorial Optimization: Networks and Matroids, (New York: Holt, Rinehart, and Winston, 1976), pp. 98-106.

[14] M. Marek-Sadowska and S. Lin, "Timing Driven Placement." *Proc. Int. Conf. on Computed-Aided Design*, 1989, pp. 94-97.

[15] A. Srinivasan, "An Algorithm for Performance-Driven Initial Placement of Small-Cell ICs," *Proc. Design Automation Conference*, June 1991, 636-639.

[16] W. J. Sun and C. Sechen, "Efficient and Effective Placement fo Very Large Circuits," *Proc. Int. Conf. on Computed-Aided Design*, 1993, pp. 170-177.

[17] S. Sutanthavibul, and E. Shragowitz, "Dynamic Prediction of Critical Paths and Nets for Constructive Timing-Driven Placement," *Proc. Design Automation Conference*, June 1991, pp. 632-635.

[18] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells," *Proc. Int. Conf. on Computed-Aided Design*, 1990, pp. 336-339.

[19] M. Terai, K. Takahashi, and K. Sato, "A New Min-Cut Placement Algorithm for Timing Assurance Layout Design Meeting Net Length Constraint," *Proc. Design Automation Conference*, June 1990, pp. 96-102.

[20] R. Tsay and J. Koehl, "An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement," *Proc. Design Automation Conference*, June 1991, pp. 620-625.

[21] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," *Management Science*, Vol. 17, July 1971, pp. 712-716.