Deriving Efficient Area and Delay Estimates by Modeling Layout Tools

Donald S. Gelosh[1] Dorothy E. Setliff
Department of Electrical Engineering
University of Pittsburgh
Pittsburgh, PA 15261
dgelosh@afit.af.mil setliff@ee.pitt.edu

Abstract— This paper presents a novel approach to deriving area and delay estimates for high level synthesis using machine learning techniques to model layout tools. This approach captures the relationships between general design features (e.g., topology, connectivity, common input, and common output) and layout concepts (e.g., relative placement). Experimentation illustrates the effectiveness of this approach for a variety of real-world designs.

I. Introduction

Design automation must produce a physical artifact with real physical characteristics. The earlier final physical characteristics are accessible the more optimal the design process and the resultant design. Unfortunately, current VLSI CAD design automation systems fail to incorporate accurate estimates in high level synthesis. The lack of accurate estimates at this level cripples the ability of the high-level synthesis to produce optimal designs in a timely manner.

McFarland, Parker, and Camposano [10] define high-level synthesis as taking a description of a design's behavior along with cost (usually area) and performance (usually delay) constraints and producing a structure that implements the behavior while satisfying the constraints. High level synthesis encourages developers to evaluate and compare possible designs. It is cost prohibitive to use actual layout tools to evaluate potential designs. Thus, area and delay estimates are instead used within synthesis.

This paper describes a method of quickly obtaining accurate area and delay estimates using machine learning to model layout tool transformations. This model captures the relationships between the physical and graphical features (e.g., size, topology, connectivity) and the relative placement of nodes as produced by the modeled layout tool. Applying the model produces a *predicted* layout. Area and delay estimates are then formed from the predicted layout. This paper illustrates how this approach is both CPU runtime cost-effective while providing a high degree of accuracy.

[1] Maj Donald S. Gelosh is now with the Air Force Institute of Technology, Wright-Patterson AFB, OH

32nd ACM/IEEE Design Automation Conference ®

The remainder of this paper is organized as follows. Section II discusses existing estimators and shows how our approach is different. Section III presents an overview of the tool modeling method and describes the general solution architecture. Section IV presents experimentation illustrating the accuracy and effectiveness of this approach. Section V reviews the contributions of this paper.

II. Previous Work

This section reviews competitive area and delay estimators. PLEST [5] provides area estimates for standard cell designs. PLEST uses two estimators: one estimates wiring space requirements for the routing channels and the other estimates the number of feedthroughs. Kurdahi and Parker claim an accuracy of within 10% of the actual areas for small designs. TELE 2.0 [14] uses a combination of constructive (partial slicing tree) and analytical (Rent's rule) approaches to estimate wire-length for standard cell designs. TELE 2.0 is both fast and accurate (7% or better), but only for small designs (less than 1800 cells). Finally, Nourani and Papachristou [11] use a non-probabilistic analytical formula to estimate standard cell layouts. They claim their estimates are within 12% for small designs (less than 2100 cells). None of these approaches consider modeling layout tools instead of analyzing the target designs or estimating layouts to gain a time and resources advantage. We show here that modeling layout tools and using the model in place of the tool is a reasonable and efficient alternative in estimating the area and delay of target designs.

III. Modeling Layout Tools

In general, machine learning attempts to generalize and quantify identified potential relationships. In this paper, the identified relationships are between design features (e.g., size, topology, connectivity) and layout concepts (e.g., relative placement). Each relationship results in a rule with an identified certainty factor attached to the rule. This certainty factor is a measure of the strict one-to-one relationship strength between a design feature and a layout concept. For example, if a rule defining the relationship between a particular design feature (e.g., input to output connectivity) and a particular layout concept (e.g., side by side relative placement) has a higher certainty factor than most, or if the certainty factor is extremely high, occurrence of input to output node connectivity results in the placement of these two nodes directly next to each

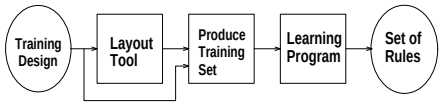

Fig. 1. Modeling Process Overview

other. Applying these rules results in a predicted layout. Analysis of this predicted layout is the basis of area and delay estimation formulation. The remainder of this section discusses the two key issues of this approach: model formulation, and use of this model to produce area and delay estimates.

A. Model Formulation

Fig. 1 illustrates the model formulation process. The target layout tool produces layouts for a set of training designs. The training designs reflect a varied cross-section of components contained in the expected target designs. The resultant layouts form a training set and serve to characterize the design-to-layout transformations of the target layout tool. Each of the training designs are typically small (<700 transistors) so producing these layouts does not consume much CPU time. In addition, this process is performed off-line and only once for a particular layout tool. The learning system takes the training set information, analyzes the relationships, and produces rules with certainty factors for each identified relationship. These rules constitute the layout tool model.

Producing the model requires first developing a training set and then using this training set to actually build the model. A training set must be sufficient (i.e., cover the operation transfer function of the layout tool). A training set is considered sufficient when it reaches a point where a substantial increase to its size does not increase the performance of the system using it [16]. In this context, a substantial increase means the addition of more training designs to the training set. The appropriate number of training designs is determined through experimentation and analysis of the results. Developing the training set is not a complex process. A good approach is to initially include at least one example of each expected target design using different styles (e.g., bit widths, algorithms, encoding schemes, etc.). Then, this large training set can be reduced until its performance on known test designs starts to fall off. At this point, the training set is considered sufficient.

The machine learning system finds relationships between features in the training designs and their corresponding layouts. Each training design instance in the training set includes a set of attribute-value pairs describing various features about the training design and a set of layout concepts that indicate relative placement of the nodes in the layout. The attribute-value pairs describe features such as the minimum area, height, and width of each node, the associated delays for rising and falling output transitions, the number of inputs, and the node's function. (All of this information is obtained from a node library as the training set is built.) The attribute-value pairs also describe the topology of the training design in relation to the node. While some of the features seem redundant, such as area when height and width are available, it is necessary to explicitly describe all features. This is because the learning system used in this approach, Rule

```
(nx1_nx2 !=)  ==> near_d
 { pos=  0.48,  neg=  0.19,  cf=  0.710
  p=31, n=1099, tp=64, tn=5872 }
```

Fig. 2. Example Rule

Learner [13], is incapable of learning relationships among the features. Rule Learner is an inductive learning system and can only learn by observation.

Because it is necessary to pair up each node with every other node in order to obtain relative placement information from the layout, node pairs are used to form the individual training instances in the training set. The features for each node pair include each node's individual set of attribute-value pairs, plus a set of relational attribute-value pairs. This includes information about which node has more area, more height, more width, more delay, more total inputs, more external inputs, and more internal inputs. Other relational features include whether or not one node has an input to, output from, or no connection to the other node, and if the two nodes share a common destination or a common source.

The relative placement information for each node pair is obtained by examining the training set layouts to determine where the nodes are located relative to each other. Each node pair has a relative placement as its concept, and the learning system tries to learn what features about the two nodes determine where they are placed relative to each other. We can use the relative placement to derive the relative distance. The relative distance determines wirelength between the logically connected nodes.

The machine learning system analyzes the training set data to determine if there are any relationships between design features found in the training set and the defined layout concepts. These relationships indicate trends and characteristics of the target layout tool. For example, if the target layout tool tends to place logically connected nodes close together, then there will be a rule describing the relationship between logically connected nodes and close placement. Each rule has a certainty factor describing the strength of the relationship. The ability to define the certainty of rules produced from a machine learning system provides an implicit ordering of rule importance and applicability.

A set of rules produced by the learning systems captures relationships between design features and layout concepts. This rule set models the layout tool, not the layout itself. These rules are of the form $< lhs > \rightarrow < rhs >$, where $< lhs >$ is a conjunction of features (attribute-value pairs) and $< rhs >$ is the layout placement concept related to the features in $< lhs >$. An example rule is shown in Fig. 2. Because the rules are based on relationships between features in the training designs and corresponding layout concepts, they reflect the characteristics of the target layout tool. When applied to a target design, these rules map features in the design to layout concepts and provide layout information that can be used to estimate area and delay.

$$cf = \frac{\frac{p}{tp}}{\frac{p}{tp} + \frac{n}{tn}} \tag{1}$$

Certainty factors (cf= 0.710 in Fig. 2) are generated by (1) where p is the number of positive examples in the training set supporting belief in the rule, tp is the total

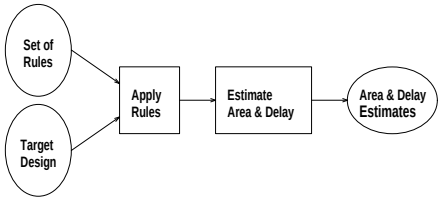

Fig. 3. Applying the Model Overview

number of examples having the same concept, n is the number of negative examples supporting disbelief in the rule, and tn is the total number of examples not having the same concept. Certainty factors should not be viewed as probabilities of a rule's truth [1]. Rather, certainty factors rank order and weight rule application.

$$cf_{comb}(X, Y) = X + Y(1 - X) \quad (2)$$

When more than one rule predicts the same concept for a node pair, the certainty factors are combined into one factor [1]. If other rules predict different concepts for the same node pair, those rules' certainty factors are combined using (2). The combination of certainty factors X and Y results in a higher certainty factor than for either X or Y alone. Combination continues until each node pair has only one certainty factor associated with it. These combined certainty factors are used to rank order the predicted concepts for that node pair.

The next section explains how the model of the target layout tool is applied to test designs to obtain area and delay estimates. The target layout tool used to illustrate this is the ArtistII [3] layout tool.

B. Formulating Area and Delay Estimates

Fig. 3 illustrates the area and delay estimate formulation process. To formulate area and delay estimates, we first apply the layout tool model to the desired target design to produce a predicted layout. This predicted layout forms the basis of the area and delay estimates. Estimates are accurate because the rules are based on how the layout tool produces layouts for the training designs. The estimates are obtained more efficiently because it takes much less time to apply the rules than to produce an actual layout.

The model of the ArtistII layout tool we used for the experiments discussed in this paper is based on a training set consisting of four training designs: a 4-bit adder, 8-bit adder, 16-bit adder, and 32-bit adder. The test designs include a 4-bit adder, 12-bit adder, 24-bit adder, 16-bit absolute value operator, and 4-bit multiplier. The test designs include adders because the training set is built from adder designs. The other test designs, the absolute value operator and multiplier, are used to show how an adder-based training set can work for other types of designs.

Model application first requires identification of the design concepts present in each of the gate pairs making up these designs, then applying the rules from the model to identify the certainty factors of the appropriate layout features. Initially, each concept is assigned a certainty factor of 0.0. Rules fire for gate pairs whose particular design concepts satisfy the rule's left-hand side. When a rule fires for a gate pair, the rule's certainty factor is combined with the predicted concept's current certainty factor for the gate pair according to (1). This combined certainty factor becomes the current certainty factor for that concept. The rule with the highest certainty factor is applied to all node pairs first, then the second most certain rule, and so on. Certainty factors are combined as necessary until each node pair has a list of predicted layout concepts rank-ordered according to their final combined certainty factor. Each of the layout concepts corresponds to a predicted relative placement of each node pair.

This rule application process is continued until all rules have been applied to all gate pairs. When this process is finished, each gate pair will have a list of predicted concepts with their final certainty factors. Each gate pair's list of concepts is then rank ordered by certainty factor in descending order. The concept with the highest certainty factor becomes the first choice for the gate pair. The concept with the next highest certainty factor becomes the second choice, and so on.

The list of gate pairs is then rank ordered in descending order according to the certainty factor of the first choice concept. Starting at the top of this list, each gate pair is assigned its first choice concept, unless that concept is unavailable due to physical limitations. Each time a concept is assigned, the physical limitation file is updated. When that concept is no longer possible for the remaining gate pairs in the predicted layout, any unassigned gate pairs in the list having that concept as their first choice must now use their second choice, if it is available. This is an example of fuzzy classification in the method. It is not necessary to assign a gate pair its first choice of relative placement concepts. For example, if the physical limitations allow only so many concepts of *near_s*, the gate pairs with the highest certainty factors for this concept should have it assigned first. When the available number of this concept runs out, then the gate pair's second choice can be assigned. Fuzzy classification gives the method flexibility when assigning concepts so physical limitations of the domain can be satisfied.

The assignment process is repeated until all gate pairs have an assigned concept. The end result is a file that lists all gate pairs with their predicted concepts. This file is rank ordered so the gate pairs near the top have been assigned concepts with the best certainty factors. The gate pairs towards the end of the list may not have been assigned their first or second choice concepts, but the certainty factors are lower towards the end due to the rank ordering. This means there is a higher probability that the first choice concepts may be incorrect as it is. The physical limitations define the bounds of the predicted layout and, thus, enhance the overall accuracy of the methodology.

B.1 Delay Estimation

The critical path determines the minimum delay through the circuit. and is a sum of all the delays between nodes on this path. The delays between nodes are calculated as a function of the driving node's intrinsic delay, the delay due to the length of wire on the driving node's output, and the load that the driven node(s) place on the driving node. The driving node's intrinsic delay comes from a physical library file. The load from the driven node(s) is a function of the number of driven nodes. This comes from the design's netlist. The relative placement of

TABLE I

DELAY ESTIMATES FOR EXAMPLE TEST DESIGNS

Design Name	Number of Gates	Actual Delay (ns)	Estimated Delay (ns)	Percent Error
add4	43	19.1	19.2	0.54%
add12	147	61.5	60.35	-1.87%
add24	303	134.0	125.06	-6.67%
abs16	208	109.0	112.36	3.08%
mult4	163	44.5	44.99	1.11%

TABLE II

AREA ESTIMATES FOR EXAMPLE TEST DESIGNS

Design Name	Number of Gates	Height Error	Width Error
add4	43	4.14%	-1.65%
add12	147	17.01%	9.88%
add24	303	-4.44%	17.90%
abs16	208	-5.38%	-3.62%
mult4	163	-1.76%	4.55%

two gates translates directly into the length of wire needed to connect them.

Table I shows the critical path delay estimates obtained for the five test designs and the percent error between these delay estimates and the actual delays from IRSIM simulations. This table shows the model performs quite well estimating delay in its final implementation for this example. This table also shows that even though the model is based on a training set using all adder designs, the model works well on other types of designs. The errors for the five test designs are all under 10%.

B.2 Area Estimation

The area estimation process looks at the list of relative placement concepts and *grows* a predicted layout. The first gate from the first gate pair in the list of predicted concepts is put in the center of the predicted layout. This becomes the *seed* gate. The second gate from the first gate pair is placed in relation to the seed gate according to the predicted concept for the gate pair. In other words, when the predicted layout is finished, this gate will be located the appropriate Manhattan distance away from the seed gate. Due to the rank ordering of the gate pairs according to certainty factors, the first gate pair in the list has an assigned concept with the best certainty factor of all. This means this gate pair has the best chance of having the correct relative placement concept. The gate placement process is continued until all gates have been placed in the appropriate positions according to their relative placement concepts.

Area estimates are generated directly from the predicted layout. Table II shows the height and width estimates obtained for the example test designs. This table shows the percent error between these estimates and the height and width values from the actual layouts. This table shows that for these test designs, the height and width estimates are within 20% of the actual values. In addition, this table shows that the model works well for area estimation on designs other than adders. These are all small designs and thus small differences yield larger than expected error rates.

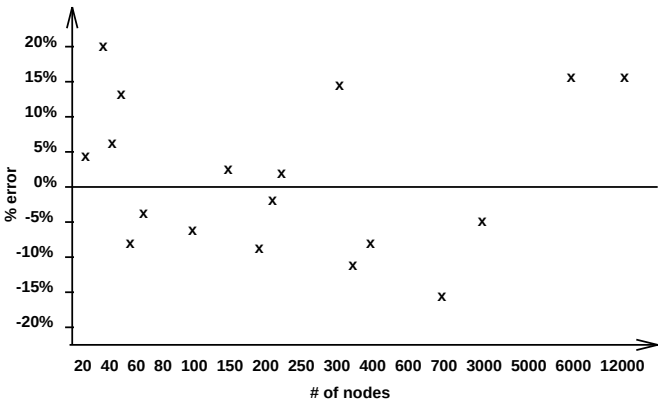

Fig. 4. Overall Error for Area Estimates

IV. RESULTS

The following results illustrate application range by showing results for ALU components as well as common synchronous benchmarks. A comparison against other approaches illustrates the competitiveness of this machine learning approach.

A. ALU Component Designs

A set of simple microprocessor ALU component designs (listed in Table III) represent a varied cross-section of different types and sizes of microprocessor ALU components. We used the TinkerTool [2] system to automatically generate VHDL descriptions of each design, and then ran Artist II to produce layouts, Magic [12] to obtain the height and width of the layout and IRSIM [15] to obtain the critical path delay. These measurements constitute the actual area and delay for error calculations. Each test design has its known actual height, width, and delay.

The following graphs show the results from applying the layout model to these designs. These graphs show percentage error vs. the wide range of design sizes found in the set of test designs. (In each of these graphs, the horizontal axis is nonlinear due to the wide range of varying design sizes.) A comparison of the estimated area and delay to the actual area and delay quantifies the overall error. For example, Fig. 4 shows the overall error for area estimates. All of these estimates are within 20% even for large designs. Fig. 5 shows a graph of the overall error for delay estimates. This graph shows the delay estimates for all of the designs are within 12%. According to Nourani and Papachristou [11], a difference of less than 10% is considered very good, but difficult to achieve. A difference of 20-30% [7] is considered to be both acceptable and achievable. Thus, these results are quite good according to these criteria.

This methodology is also much faster than producing an actual layout. Table III shows the total time to derive area and delay estimates is much less than producing an actual layout. Combining these execution time results with the area and delay estimate results shows that this method of modeling layout tools does produce accurate estimates very quickly.

Layouts for all of the test designs except for the last three were produced by ArtistII using 10,000 iterations.

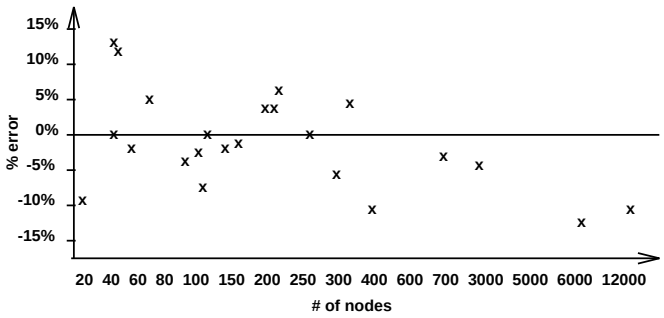

Fig. 5. Overall Error for Delay Estimates

TABLE III
EXECUTION TIME COMPARISON (CPU SEC)

Design Name	Number of Gates	Total Time	ArtistII Time
add4	43	0.49	840
add8	95	0.96	2278
add12	147	1.70	5385
add16	199	2.82	8278
add24	303	5.82	11886
add32	407	10.71	18735
abs16	208	2.71	6676
inc4	23	0.61	405
inc8	51	1.15	980
inc16	107	0.97	3408
sub4	48	0.48	828
sub8	104	1.18	2374
sub16	216	3.67	7788
addsub4	64	0.80	1398
addsub8	132	1.44	4896
addsub16	268	4.24	10224
mult4	163	2.22	4106
mult8	711	29.03	47696
mult16	2959	442.57	28746
big	6264	2215.44	65493
realbig	12528	7605.73	359289

Due to their large size, the last three designs in the table, *mult16*, *big*, and *realbig* required a reduced number of iterations (100) in order for ArtistII to produce a layout at all! (In order to keep the comparisons valid, we built a different training set using the same training designs that models ArtistII using only 100 iterations and used the resultant model to obtain the execution times for these three large designs.)

B. Comparison to Other Area and Delay Estimators

Table IV shows how our area and delay estimation method compares to other estimators. A comparison of results from this estimation method to estimators from Nourani [11] and in the TELE 2.0 [14] and PLEST [5] systems show that this modeling method is very competitive in area and delay estimation. The estimation method has the advantage of being able to aggressively handle larger designs as shown in Table III.

TABLE IV
COMPARISON TO OTHER ESTIMATORS

System Name	Design Name	Dim.	Their Error	Our Error
(Nourani)	mult4	ht	3.09%	0.77%
		wd	4.22%	1.83%
	mult8	ht	4.69%	-1.60%
		wd	6.87%	-15.93%
TELE 2.0	mult16	area	-13.91%	-4.85%
		del	18.00%	-6.30%
	add32	area	9.62%	-9.37%
		del	6.00%	-3.08%
PLEST	mult8	area	5.8%	1.05%
	add16	area	10.6%	6.51%
	mult16	area	3.1%	-4.85%

TABLE V
AREA ESTIMATES FOR BENCHMARK DESIGNS

Design Name	Number of Gates	Height Error	Width Error
DIV16	2085	-16.73%	0.02%
GCD32	3669	5.51%	-12.82%
DIV32	4177	14.42%	9.64%

C. Benchmark Results

This section focuses on benchmark designs [8]. These benchmark designs each contain more than 2000 gates, so again, a reduced number of iterations (100) of the ArtistII model was necessary. Table V shows the height and width errors found using this method. Table VI shows the actual delay obtained from an IRSIM simulation of the benchmark design and the critical path delay derived from information produced by applying our model of ArtistII to the benchmark design. These results show the accuracy of this approach even for complex synchronous designs.

V. CONCLUSIONS

This paper presents a methodology using machine learning to model layout tools, rather than the layout itself. Using this model in place of the layout tool quickly produces accurate area and delay estimates. Through experimentation with a number of training designs and test designs, and two target layout tools, this paper demonstrates the validity of this modeling method and the accuracy of the area and delay estimates. This paper also demonstrates the scalability of this approach by producing estimates within 10% for designs up to 12,000 gates. The modeling method is orders of magnitude faster and includes estimating area and critical path delay. A comparison of estimation results from both of the models to other area and delay estimators illustrate the competitiveness of this approach.

This methodology aids the design automation process

TABLE VI
DELAY ESTIMATES FOR BENCHMARK DESIGNS

Design Name	Number of Gates	Actual Delay (ns)	Estimated Delay (ns)	Percent Error
DIV16	2085	231.0	199.71	-13.55%
GCD32	3669	331.7	327.78	-1.18%
DIV32	4177	344.7	336.15	-2.48%

by decreasing the overall design time while increasing the effectiveness of high-level decision making ability. It does this by modeling the layout tool being used, and not the design itself. This model is used in place of the layout tool to provide the necessary layout information based on the design. This layout information along with graph theory principles is used to derive accurate area and delay estimates in a timely fashion.

Acknowledgment

We are very grateful to Professor Steven Levitan and his colleagues at the University of Pittsburgh for their assistance. Our thanks also to the reviewers of this paper for their helpful suggestions and advice.

References

[1] Buchanan, B. and Shortliffe, E. *Rule-Based Expert Systems*, (Reading: Addison-Wesley, 1984) pp. 247-262.

[2] Hsieh, Yee-Wing, "Architectural Synthesis Via VHDL" (M.S. thesis, the University of Pittsburgh, 1992).

[3] Irwin, M.J. and Owens, R.M., "A Comparison of Four Two-Dimensional Gate Matrix Layout Tools," *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 698-701, 1989.

[4] Kang, S., "Linear Ordering and Application to Placement," *Proceedings of the 20th IEEE/ACM Design Automation Conference*, pp. 457-464, 1983.

[5] Kurdahi, F.J. and Parker, A.C., "Techniques for Area Estimation of VLSI Layouts," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 1, (Jan. 1989), pp. 81-92.

[6] Landman, B. and Russo, R., "On a Pin Versus Block Relationship for Partition of Logic Graphs," *IEEE Transactions on Computers*, vol. C-20, pg. 1469, 1971.

[7] Personal communication with Steven P. Levitan, Assistant Professor, Department of Electrical Engineering, University of Pittsburgh, PA., September 6, 1993.

[8] MCNC, Center for Microelectronic Systems Technologies, 3021 Cornwallis Road, P.O. Box 12889, Research Triangle Park, N.C. 27709.

[9] McFarland, M.C., Parker, A.C., and Camposano, R., "Tutorial on High-level Synthesis," *Proceedings of the 25th ACM/IEEE Design Automation Conference*, 1988, pp. 330-336.

[10] McFarland, M.C., Parker, A.C., and Camposano, R., "The High-Level Synthesis of Digital Systems," *Proceedings of the IEEE*, Vol. 78, No. 2 (February 1990), pp. 301-317.

[11] Nourani, M. and Papachristou, C., "A Layout Estimation Algorithm for RTL Datapaths," *Proceedings of the 30th ACM/IEEE Design Automation Conference*, 1993, pp. 285-291.

[12] Ousterhout, J., Hamachi, G., Mayo, R., Scott, W., and Taylor, G., "Magic: A VLSI Layout System," *Proceedings of the 21st Design Automation Conference*, pg. 152, 1984.

[13] Provost, F.J., "Policies for the Selection of Bias in Inductive Machine Learning" (Ph.D. Thesis, University of Pittsburgh, 1992).

[14] Ramachandran, C. and Kurdahi, F.J., "TELE: A Timing Evaluator Using Layout Estimation for High Level Applications," *Proceedings of the European Design Automation Conference*, 1992.

[15] Terman, C.J. "Simulation Tools for Digital LSI Design" (Ph.D. thesis MIT Laboratory of Technology for Computer Science, 1983).

[16] Weiss, Sholom M. and Kulikowski, Casimir A., *Computer Systems That Learn*, (San Mateo: Morgan Kaufmann, 1991).