MASSACHUSETTS INTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-556
September 1989

Optimal Layout Via Boolean Satisfiability

Srinivas Devadas

Abstract

Most optimization problems in layout have been shown to be NP-complete, resulting in researchers abandoning the search for optimum solutions even for small-scale problem instances.

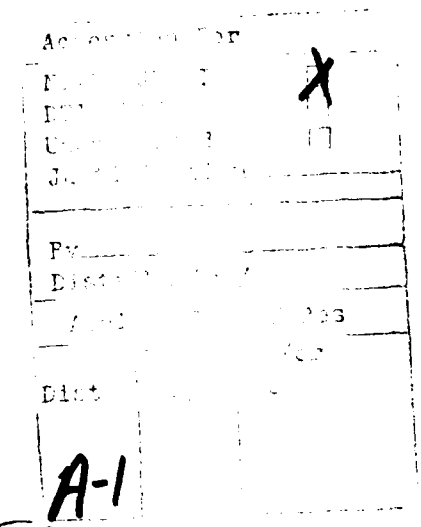
In this paper we transform various NP-complete problems in layout, namely two- and multi-layer dogleg routing, two-way partitioning, one-dimensional and two-dimensional placement, into *Boolean satisfiability* problems. The transformations are efficient in that the number of inputs to the Boolean function, for which we have to find a satisfying assignment, *only grows linearly or quasi-linearly* with the layout problem size. These transformations also produce a *minimal-sized* Boolean function, in order to speed up satisfiability check performance.

We apply sophisticated test generation and logic verification strategies that can be used to check for Boolean function satisfiability to these layout problems. We present experimental results which indicate that *problems of significant size can be solved optimally* using this approach. This approach to optimal layout is considerably more efficient than exhaustive search.

Further, we show that this approach to layout optimization offers an elegant means of representing and searching *the entire space of feasible solutions* in an attempt to optimize a complex cost function with associated constraints.

Microsystems Technology Laboratories
Massachusetts Institute of Technology
Cambridge Massachusetts 02139

Room 39-321
Telephone
(617) 253-0292



Acknowledgements

To be presented at the *International Conference of Computer Aided Design (ICCAD '89)* in November 1989. This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-87-K-0825.

Author Information

Devadas: Department of Electrical Engineering and Computer Science, Room 36-848, MIT, Cambridge, MA 02139. (617) 253-0454.

Optimal Layout Via Boolean Satisfiability

Srinivas Devadas

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge

Abstract

Most optimization problems in layout have been shown to be NP-complete, resulting in researchers abandoning the search for optimum solutions even for small-scale problem instances.

In this paper, we transform various NP-complete problems in layout, namely two and multi-layer dogleg channel routing, two-way partitioning, one-dimensional and two-dimensional placement into *Boolean satisfiability* problems. The transformations are efficient in that the number of inputs to the Boolean function, for which we have to find a satisfying assignment, *only grows linearly or quasi-linearly* with the layout problem size. These transformations also produce a *minimal-sized* Boolean function, in order to speed up satisfiability check performance.

We apply sophisticated test generation and logic verification strategies that can be used to check for Boolean function satisfiability to these layout problems. We present experimental results which indicate that *problems of significant size can be solved optimally* using this approach. This approach to optimal layout is considerably more efficient than exhaustive search.

Further, we show that this approach to layout optimization offers an elegant means of representing and searching *the entire space of feasible solutions* in an attempt to optimize a complex cost function with associated constraints.

1 Introduction

The area of layout design and synthesis is rich in optimization problems. Placement problems involve finding locations for various modules so as *to minimize a given objective function*, which reflects the interconnect complexity of the placed modules. Given a placement of modules, the problem of routing is to make the required connections in minimum area.

Unfortunately, most placement and routing problems have been shown to be NP-complete. This has led researchers into abandoning the search for optimum solutions even for small-scale problem instances. Other than Asano's optimum one-dimensional placement algorithm [1] and optimum bipartite PLA folding techniques [5], there has been a dearth of algorithms that guarantee global optimality for layout problems.

With computers becoming consistently and significantly faster at each new generation, it is our belief that NP-complete optimization problems, of a significant size, can be solved in an optimal way, with reasonable CPU time expenditure. This belief stems from the fact that even though the worst-case performance of exact algorithms for NP-complete problems grows exponentially with problem size, in practice quite a few of these problems have been shown to be amenable to an "intelligent" exact solution method, e.g. branch and bound techniques, even for large-scale problem instances. By an intelligent method, we mean a method that prunes the search space without sacrificing optimality by exploiting the nature of the problem and the structure of the problem instance. Trivial exhaustive search methods have a worst-case *and* an empirical behavior that is exponentially related to problem scale.

Real-life, large-scale instances of several NP-complete logic synthesis problems, notably ones of two-level Boolean minimization, Boolean tautology and satisfiability, have been solved using exact, albeit very sophisticated algorithms. For details, the reader is referred to [3] [4] [8] [10].

In this paper, we transform various NP-complete problems in layout, namely two and multi-layer dogleg channel routing, two-way partitioning, one-dimensional and two-dimensional placement into *Boolean satisfiability* problems. The transformations are efficient in that the number of inputs to the Boolean function, for which we have to find a satisfying assignment, *only grows linearly or quasi-linearly*[1] with the layout problem size. The number of gates in the Boolean function, resulting from this transformation, is also *minimal*. These transformations allow us to apply sophisticated test generation and logic verification strategies that can be used to check for Boolean function satisfiability to layout problems. We have obtained experimental results which indicate that routing and placement problems of significant size can be solved optimally using this approach. A trivial exhaustive search method is not feasible for anything but very small problems. Thus, this approach to optimal layout is considerably more efficient than exhaustive search.

In Section 2, the algorithms we use for the Boolean satisfiability check are briefly described. Transformations for routing and placement problems to Boolean function satisfiability are described in Sections 3 and 4. We describe implementation details and give preliminary experimental results in Section 5.

[1] We use the term quasi-linear to signify a $n\log(n)$ growth.

2 Checking for Boolean Satisfiability

There are several ways that a Boolean function can be checked for satisfiability. It is not our purpose here to review the extensive literature on the subject and hence we focus on two particular techniques that we experimented with.

One method of checking two logic functions for equivalence [3] transforms the given logic functions *into canonical representations* known as Binary Decision Diagrams (BDDs). Because BDDs represent canonical graphs of logic functions, checking logic functions for equivalence, *given their BDDs*, becomes merely a graph isomorphism check, which can be performed in time linear in the size of the BDDs. Also, a logic function is satisfiable if and only if its BDD is not the trivial 0 BDD. The construction of a BDD and its reduction to a canonical form has worst-case exponential complexity, *which may require unreasonable amounts* of CPU time and memory. However, BDDs have been constructed in several cases, for functions with over a hundred inputs and a thousand gates, implying that satisfiability checking for large functions is possible via this method.

Another technique [8] uses test generation algorithms like those in [6] to enumerate, in sum-of-products form, the ON-set or the OFF-set of the given logic function. If the *sum-of-product form* for the ON-set of a function is null, it means that the function is not satisfiable, else it is satisfiable. This method does not require large amounts of memory, but the CPU time requirements can grow exponentially with the number of inputs in the problem instance.

3 Channel Routing

3.1 Introduction

The routing problem in integrated circuit (IC) layout is to realize a specified interconnection among modules in as small an area as possible. Channel routing involves routing a specified netlist between two rows of terminals across a channel.

Channel routing has been the subject of extensive research. The goal of a channel router is to route a particular channel using a minimum *number* of horizontal tracks. Several algorithms have been proposed for two-layer channel routing (e.g. [9], [12]) and more recently, for multi-layer channel routing (e.g. [2]). These algorithms perform very well in practice [2], but are heuristic and offer *limited guarantees* as to the quality of the final route.

The wiring model used in channel routers assumes that each layer runs in a particular direction, i.e. either horizontal or vertical. This constraint is sometimes relaxed (as in [2] and [9]), in order to obtain better solutions.

3.2 Channel Routing Basics

Given N nets to be routed, we have to place these N nets on a minimum number of tracks. There are two distinct types of constraints that have to be accommodated when routing nets. If we define the **interval** of a net i as $I(i) = [l_i, r_i]$, where l_i (r_i) is the position of the leftmost

[2] These algorithms achieve the lower bound on the required number of tracks for problem instances in many cases.

(rightmost) terminal the net has to be connected to, then it is clear that nets with intersecting intervals cannot be placed on the same row.

A second type of constraint is related to the wiring model used and deals with column-wise relationships between nets. If a net i is connected to the top terminal at column c and net j is connected to the bottom terminal at the same column c, then we require net i to be placed *above* net j, in order to make the vertical connections. One can derive a Vertical Constraint Graph (VCG) for any given channel, that represents all vertical constraints between nets.

In the simplest form of routing, each net is realized as a unbroken horizontal segment. A degree of freedom that can be utilized to advantage lies in *doglegging* a net, i.e. breaking a net into (two or more) sub-nets at its terminal positions. Doglegging can reduce the number of tracks required to make a route.

3.3 Two-Layer Channel Routing Via Boolean Satisfiability

Given a channel, the following procedure constructs a Boolean function such that if the function is satisfiable, then it implies that the channel can be routed in D tracks. A satisfying assignment for the Boolean function can be trivially mapped onto a route in D tracks. If the Boolean function is not satisfiable, it implies that no route in $\leq D$ tracks exists under the chosen wiring model. Szymanski [11] transformed the problem of 3-Satisfiability into one of channel routing in order to prove that channel routing was NP-complete. Here, we are interested in the reverse transformation.

1. Each net i, has $P = \lceil log_2(D) \rceil$ Boolean variables associated with it, namely v_{i1}, v_{i2}, .. v_{iP}. These variables are bit vectors that correspond to the track number onto which the net will be placed. These variables are inputs to the logic function that will be constructed.

2. For each net pair i, j, if $I(i) \cap I(j) \neq \phi$, generate the logic functions shown below:

$$v_{i1} \oplus v_{j1} + v_{i2} \oplus v_{j2} + .. + v_{iP} \oplus v_{jP} \qquad (1)$$

where $\oplus$ is the exclusive-or Boolean operator. The above equations represent the constraints that nets with intersecting intervals cannot be placed on the same row.

3. If $D < 2^P$, then arbitrarily pick $2^P - D$ distinct bit-vectors of length P. These bit-vectors will correspond to unused rows. We do not want any nets on these rows and hence we generate the logic below, for each net and each unused bit-vector $[c_1\ c_2\ ..\ c_P]$.

$$v_{i1} \oplus c_1 + v_{i2} \oplus c_2 + .. + v_{iP} \oplus c_P \qquad (2)$$

It should be noted that c_1, .. c_P are constants with values 0 or 1.

4. Remove redundant edges from the VCG. For instance, if there are edges between nets i and j, i and k and j and k, the edge between i and k is removed. For each edge between net i and net j in the VCG, generate the logic corresponding to:

$$(v_{i1}, v_{i2}, .. v_{iP}) > (v_{j1}, v_{j2}, .. v_{jP}) \qquad (3)$$

For instance, when $P = 2$, the logic will be

$$v_{i1}.v_{i2}.\overline{v}_{j2} + v_{i2}.\overline{v}_{j1}.\overline{v}_{j2} + v_{i1}.\overline{v}_{j1} \qquad (4)$$

5. Construct a Boolean function F that is the conjunction of Eqns. 1, Eqns. 2 and Eqns. 3, with inputs v_{ik}, $\forall\ i,\ k$. Check F for satisfiability.

A bit-vector corresponds to the row number for each net. We have D possible bit-vectors that can be assigned to the nets. The logic generated in the above procedure has a one-to-one correspondence with the constraints described in Section 3.2.

Theorem 3.1 : *A route using $\leq D$ tracks exists if and only if the Boolean function constructed by the above procedure is satisfiable.*

3.4 Finding An Optimal Route

The procedure described allows us to determine whether or not a channel is routable in D tracks. To find an optimal route, we repeatedly apply satisfiability checks as described below.

1. Given a channel, we can obtain lower bounds on the number of tracks required to route it on the basis of the maximum density across all columns in the channel as well the longest path [3] in the VCG. We first attempt to find a route in a number of tracks equal to the maximum of these two bounds, namely D.

2. Try to find a satisfying assignment for the Boolean function, corresponding to routing the channel in D tracks. If the satisfiability checker does not terminate within a specified amount of time, discontinue the search and go to Step 3. Else, if a satisfying assignment is found, we have found a route in D tracks. If a satisfying assignment is not possible, go to Step 3.

3. $D = D + 1$. Go to Step 2.

If all satisfiability checks terminate, i.e. are not discontinued, then the procedure above guarantees an optimal routing. All the nets or a subset of nets can be doglegged initially, to break cycles in the VCG or to reduce the longest path.

3.5 Improving Satisfiability Check Performance

It is of great importance that the Boolean function produced via the logic generation procedure, F, be as small as possible. The number of inputs to the Boolean function is $N \times \lceil log_2(D) \rceil$, where N is the number of nets in the channel and D is the number of targeted tracks. The transformation is efficient in that the number of inputs to F only grows linearly with the number of nets. The amount of logic generated is dependent on the problem instance. If a large number of nets overlap, we will have a large amount of logic corresponding to Eqns. 1.

Since the $>$ operator implies the $\neq$ operator, if two overlapping nets have a directed path between each other in the VCG, we do not have to generate logic corresponding to the Eqns. 1 for these nets.

In Step 3 of the logic generation procedure of Section 3.3, redundant edges in the VCG are removed, so as to minimize the amount of logic generated. If these redundant edges are not removed, they will correspond to redundant logic in F.

Eqns. 2 can also be simplified, if a large number of unused bit-vectors exist. For instance, if 1111 and 1110 are two unused bit-vectors, we can merge them into 111– and generate logic for only this bit-vector. We minimize the set of unused bit-vectors using a two-level Boolean minimizer like ESPRESSO [10], so as to reduce the number of Eqns. 2. This has to only be done once for each satisfiability check.

An alternate representation of Eqns. 2 can be used if there are a large number of unused codes and/or nets. We pick the $2^P - D$ bit-vectors with the largest decimal value. Instead of generating inequality logic, we generate logic corresponding to the rows of nets being *less* than a compacted form (cubes) of these bit vectors. The advantage of this representation is that we can exploit the transitivity of the $<$ operator, and generate logic only for the nets at the highest level in the VCG.

Eqns. 3 are stored in a compact form for different values for p. Eqn. 4 is the minimum representation for $p = 2$. Minimum representations for $p > 2$ are stored as logic macros and instantiated with different input and output variables whenever necessary.

The VCG is leveled from the top to bottom as well as from the bottom to the top. The level from the top and bottom for each net represent the range of row positions that a net can occupy. If lt_i (lb_i) is the level from the top (bottom) for net i and there are R rows, then the row of a net has to lie in the closed interval $[R - lt_i + 1,\ lb_i]$. This limits the values that the Boolean variables v_{ij} of net i can take. We find the row range for each net and set the appropriate variables to constant values of 0 or 1, *decreasing the number of input variables in the Boolean function to be checked for satisfiability*. For instance, if $R = 8$ and the range of a net k is $[4,\ 5]$, it means that the net has to lie on rows 100 or 101 and therefore $v_{k1} \equiv 1$ and $v_{k2} \equiv 0$ (The number of inputs has effectively been reduced by 2).

These steps are vitally essential to solving large problems, as is indicated in Section 5.

3.6 Representing Feasible Solutions

Constructing a Binary Decision Diagram (BDD) representing the Boolean function corresponding to the routability of a channel in D tracks, is a means of *implicitly* but exhaustively searching *all* possible feasible solutions to routing the channel in D tracks.

The BDD offers a compact representation of the ON/OFF-sets of a Boolean function. Each minterm in the ON-set corresponds to *distinct* feasible route of the channel in D tracks. However, while the number of minterms in the ON-set can be exponential in the number of inputs to the function, the size of the BDD itself can be small – this is because

[3] The length of the longest path can be changed via doglegging, but not the maximum density.

each minterm corresponds to a *path* in the BDD from the root to a leaf. In fact, several minterms, forming a cube, can correspond to a single path. The number of paths may be extremely large, growing exponentially with the number of edges in the BDD.

If one is targeting a complex cost function, including a relatively simple minimum-track solution, then one can simply search this compact representation of the entire space of feasible solutions to select the best solution under the cost function. A example would be *timing-driven* routing, where the length of certain nets needs to be small. Minimum-track solutions can be compared against each other to find the one with the desired property.

3.7 Extensions to Multi-Layer Channel Routing and Other Wiring Models

The procedure can be extended to the multi-layer routing case, for a given layer-direction configuration. For example, if we have three layers, one can perform a VHV route or a HVH route. For simplicity in description, we will indicate in the sequel how optimal VHV and HVH routes can be found via Boolean satisfiability checking. However, the logic generation procedure can be generalized to > 3 layers.

In the case of VHV routing, there are no vertical constraints between nets and the horizontal segments can only be on one layer. Therefore, the logic generation procedure of Section 3.3, is unchanged, except that Eqns. 3 are not generated; the VCG is ignored.

The HVH case is more complicated. The lower bound on HVH routing is $d/2$ where d is the maximum density across the channel. We can place a net on two possible horizontal layers. We add a variable l_i to each net i that corresponds to the layer that the net is on. $l_i = 1$ implies that net i is layer 1 and $l_i = 0$ implies that net i is on layer 2. Two nets with intersecting intervals have to be on different rows or on different layers. We can write

$$v_{i1} \oplus v_{j1} + \ldots + v_{iP} \oplus v_{jP} + l_i \oplus l_j$$

in correspondence to Eqns. 1. Eqns. 3 are unchanged for the HVH case, since vertical constraints have to be obeyed. Unused bit-vectors also have to be handled via Eqns. 2.

If the pitch of the two layers is different, then we choose the track count such that we have $d_1 + d_2 = d$ horizontal tracks available, where d_1 (d_2) is the number of tracks on layer 1 (2). The number of variables we require per net (other than the layer variable) is $p = \lceil log_2(MAX(d_1, d_2)) \rceil$. We will have $2^p - d_1$ unused bit-vectors for layer 1 and $2^p - d_2$ unused bit-vectors for layer 2.

With four or more layers, we will have vertical constraints between nets if they are on the same layer, but not if they are on different layers.

A technique which has been used successfully to route channels with cyclic VCGs [9] is to break cycles in the VCG, route the channel according to the acyclic VCG and use a maze router to make the connections corresponding to the edges that were initially removed from the VCG. The maze router uses a more flexible wiring model and is efficient for the small number of connections that can be made. We can use the technique described in this section to perform an optimal route for the acyclic VCG, rather than the heuristic strategy used in [9].

4 Partitioning and Placement

4.1 Introduction

Several placement problems can be efficiently transformed into Boolean satisfiability problems and solved exactly. In this section, we describe transformations for two-way partitioning, as well as one-dimensional and two-dimensional placement problems. All these transformations have the property that the number of inputs to the corresponding Boolean function grows quasi-linearly with problem size.

4.2 Two-way Partitioning

The problem addressed here is the classical graph partitioning problem. Given a graph $G(V, E)$ with weights on its edges, partition the nodes of G into two subgraphs A and B, such that the cumulative weight of edges across the partition is minimum. Heuristic algorithms have been proposed for this problem (e.g. [7]). Here, we are concerned with a procedure that guarantees the minimum solution. In the sequel, we will deal with uniform two-way partitioning, where $||A|| = ||B||$, for simplicity. However, the procedure can easily be generalized to the the non-uniform case.

We can generate a Boolean function such that if and only if the function is satisfiable, will there exist a partition of the nodes, $V \in G$, that has a cost $\leq C$. If one can obtain a lower bound, L_C, on the cost of the optimum partition, then we can use the strategy of the previous section, in progressively increasing the targeted cost, C^T, till we find a function that is satisfiable (that corresponds to an optimum partition). However, good lower bounds do not exist for the partitioning problem unlike channel routing. Therefore, we find a good upper bound, U_C, using the heuristic technique of [7] and progressively *decrease* the targeted cost, C^T, beginning from U_C, till we encounter a logic function that is *not* satisfiable. Then, $C^T + 1$ is the cost of the optimum partition. The decrement in C^T is related to problem size; for small problems C^T is decremented by 1 at each pass, for larger problem instances, a decrement greater than unity is used.

The logic generation procedure for two-way uniform partitioning is described below. The procedure receives the graph and a targeted cost, C^T, as inputs. N is the number of nodes in the graph.

1. Each node $i \in G$, has $P = \lceil log_2(N) \rceil$ Boolean variables associated with it, namely $v_{i1}, v_{i2}, \ldots v_{iP}$. These variables are bit vectors whose P-th bit correspond to the partition in which the node will be placed. There are two distinct partitions corresponding the 0/1 values for each of the v_{iP}. These variables are inputs to the logic function that will be constructed.

2. For each node pair i, j generate the logic functions shown below:

$$v_{i1} \oplus v_{j1} + v_{i2} \oplus v_{j2} + \ldots + v_{iP} \oplus v_{jP} \quad (5)$$

where $\oplus$ is the exclusive-or Boolean operator. The above equations represent, in a compact way, the constraint of uniform partitioning, namely, that $||A|| = ||B||$. We will have $N/2$ nodes whose P-th bit has the value 1 and $N/2$ nodes whose P-th bit has the value 0.

3. If $N < 2^P$, then pick $2^P - N$ distinct *sequential*[4] bit vectors of length P. These bit-vectors will correspond to unused codes for the nodes.

$$v_{i1} \oplus c_1 + v_{i2} \oplus c_2 + \ldots + v_{iP} \oplus c_P \quad (6)$$

As in the routing case, $c_1, \ldots c_P$ are constants with values 0 or 1.

4. For each edge in the graph, we generate logic to check if the edge is a crossing edge or not. If so, the weight of the edge is added to the cost C. We construct the Boolean function

$$C = \sum_{i=1}^{||E||} wt(e_i) * (v_{iP} \oplus v_{jP}) \quad (7)$$

where k and l are the nodes connected to edge $e_i \in I$, $wt(e_i)$ is the weight of edge e_i. The $*$ operator represents a simple form of multiplication, where the second operand is a single bit operand, that can take on the values 0 or 1.

5. Represent the constraint that the cost has to be less than or equal to the targeted cost C^T.

$$C \leq C^T \quad (8)$$

This is the complement of the $>$ operator described in the previous section.

6. Construct a Boolean function I that is the conjunction of Eqns. 5, Eqns. 6 and Eqns. 8, with inputs v_{ik}, $\forall$ i, k. Check I for satisfiability.

Theorem 4.1 : *A partition with cost $\leq C^T$ exists if and only if the Boolean function constructed by the above procedure is satisfiable.*

4.3 One-Dimensional and Two-Dimensional Placement

The problem of one-dimensional placement of gates, so as to minimize the total net length (or minimize some of the weighted sum of net lengths or minimization with constraints on the net lengths) has a very similar formulation to that above. We have $\lceil log_2(N) \rceil$ variables for each net, corresponding to the position the net can assume. The cost of a placement is represented by the sum of the net lengths, much like in Eqn. 7.

In two-dimensional placement, under a given aspect ratio, we will have two sets of variables corresponding to the X and Y coordinates for each net. The cost is calculated as a function of both sets of variables.

These problems are intrinsically more difficult than routing or partitioning problems. We feel, at this stage, that satisfiability checkers have to develop further, so as to be able to solve large two-dimensional placement problems. However, highly constrained placement problems can be solved optimally, even now, since constraints preclude the legality of a large space of solutions.

[4] By sequential, we mean monotonically increasing in decimal value.

Ex	#cols	#nets	density
nasty5.5	17	11	7
nasty7.7	21	13	9
nasty9.7	21	11	7
3a	45	30	15
3b	62	47	17

Table 1: Statistics of Channels

Ex	NO-DOGLEG			DOGLEG		
	#tra	#sat	CPU time	#tra	#sat	CPU time
nasty5.5	9	3	2.1m	8	2	2.3m
nasty7.7	11	3	2.9m	10	2	3.9m
nasty9.7	10	3	4.1m	9	2	4.8m
3a	16	2	23m	15	1	19m
3b	19	3	36m	18	2	21m

Table 2: Optimal Two-Layer Channel Routing

5 Results

In this section, we present preliminary experimental results using the procedures described in the previous sections. In all cases, the Binary Decision Diagram [3] method for satisfiability checking was used, since its performance was superior to the test generation method, for the kinds of functions that we encountered.

In Table 1, the statistics of the examples used are given. The number of columns (#cols), nets (#nets) and maximum density (density) are indicated for each example. We present optimal two-layer routing results in Table 2. Results without doglegging are given under the column NO-DOGLEG. The optimum number of tracks (#tra), number of Boolean satisfiability checks (#sat) and the total CPU time for all the satisfiability checks (CPU time) are indicated for each example. Medium-sized channels are amenable to optimum routing. But for the smallest example, none of the others can be solved via an exhaustive search method. The CPU times are all on a VAX 11/8800. The CPU times for logic generation are negligible in comparison to the satisfiability checking times, and hence are not given.

We present results for 3-layer routing in Table 3. Results for both the VHV and HVH wiring models are given. No doglegging was performed in these experiments. Though the 3-layer routing problem is more complicated, still problems of significant size can be solved exactly.

Finally, we present results for two-way partitioning in Table 4. The number of nodes to be partitioned (#nodes), the number of nets connecting them (#nets), the cost of the optimum partition (cost), the number of Boolean satisfiability checks (#sat) and the CPU time required (CPU time) for all the checks are indicated. The heuristic algorithm of [7] was used to obtain an upper bound on the cost for each problem and logic functions corresponding to gradually decreasing costs were generated, until a logic function that was not satisfiable was encountered.

6 Conclusions

In this paper, we have shown how various problems in layout, namely, two and multi-layer dogleg channel routing, two-way partitioning, one-dimensional and two-dimensional placement can be transformed efficiently into Boolean satisfiability problems and solved optimally using sophisticated test generation and logic verification techniques. These

Ex	VHV Routing			HVH Routing		
	#tra	#sat	CPU time	#tra	#sat	CPU time
nasty5.5	7	1	21s	5	1	44s
nasty7.7	9	1	17s	5	1	57s
nasty9.7	7	1	36s	5	1	74s
3a	15	1	11m	8	1	13m
3b	17	1	12m	9	1	21m

Table 3: Optimal Multi-Layer Channel Routing

Ex	#nodes	#nets	#sat	start cost	Min. cost	CPU time
ex1	7	34	6	36	31	53s
ex2	11	54	5	42	38	4m
ex3	13	61	10	55	46	9m
ex4	21	141	11	10	0	16m
ex5	32	257	12	28	17	11m

Table 4: Optimal Two-Way Partitioning

transformations are efficient in that the number of inputs in the Boolean function, we have to find a satisfying assignment for, only grows linearly or quasi-linearly with the layout problem size. The number of gates in the Boolean function is minimized during the generation of logic, so as to speed up satisfiability check performance.

Experimental results indicate that this method of optimal layout is viable for medium-sized problems and is much more efficient than exhaustive search.

An attractive feature of using this approach is that the entire space of feasible solutions can be represented in a compact way, facilitating the search for optimal solutions under complex cost functions and associated constraints.

7 Acknowledgements

The interesting discussions with Tony Ma, Sharad Malik and Richard Newton on layout problems and Boolean satisfiability are acknowledged. This research was supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

References

[1] T. Asano. An Optimum Gate Placement Algorithm for MOS One-Dimensional Arrays. In *Journal of Digital Systems*, pages 1-25, January 1982.

[2] D. Braun, J. Burns, F. Romeo, A. Sangiovanni-Vincentelli, K. Mayaram, S. Devadas, and H-K. T. Ma. Techniques for Multi-Layer Channel Routing. In *IEEE Transactions on CAD*, pages 698-712, June 1988.

[3] R. Bryant. Symbolic Verification of MOS Circuits. In *Proc. of 1985 Chapel Hill Conference on VLSI*, pages 419-438, December 1985.

[4] M. Dagenais, V. K. Agarwal, and N. Rumin. McBOOLE: A Procedure for Exact Boolean Minimization. In *IEEE Transactions on CAD*, pages 229-237, January 1986.

[5] P. Egan and C. L. Liu. Optimal Bipartite Folding of a PLA. In *IEEE Transactions on CAD*, pages 191-198, July 1984.

[6] P. Goel. An Implicit Enumeration Algorithm to generate tests for combinational logic circuits. In *IEEE Transactions on Computers*, pages 215-222, March 1981.

[7] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. In *The Bell System Technical Journal*, pages 291-307, February 1970.

[8] H-K. T. Ma, S. Devadas, R-S. Wei, and A. Sangiovanni-Vincentelli. Logic verification algorithms and their parallel implementation. In *IEEE Transactions on CAD*, pages 181-189, February 1989.

[9] J. Reed, A. Sangiovanni-Vincentelli, and M. Santamauro. A new symbolic channel router: yacr2. In *IEEE Transactions on CAD*, pages 208-219, July 1985.

[10] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. In *IEEE Transactions on CAD*, pages 727-751, September 1987.

[11] T. Szymanski. Dogleg Channel Routing is NP Complete. In *IEEE Transactions on CAD*, pages 31-40, January 1985.

[12] T. Yoshimura and E. S. Kuh. Efficient Algorithms for Channel Routing. In *IEEE Transactions on CAD*, pages 25-35, January 1982.