

Formulation of Static Circuit Optimization with Reduced Size, Degeneracy and Redundancy by Timing Graph Manipulation

Chandu Visweswariah

Andrew R. Conn

IBM Thomas J. Watson Research Center

Route 134 and Taconic

Yorktown Heights, NY 10598

{chandu,arconn}@watson.ibm.com

Abstract

Static circuit optimization implies sizing of transistors and wires on a static timing basis, taking into account all paths through a circuit. Previous methods of formulating static circuit optimization produce problem statements that are very large and contain inherent redundancy and degeneracy. In this paper, a method of manipulating the timing formulation is presented which produces a dramatically more compact optimization problem, and reduces redundancy and degeneracy. The circuit optimization is therefore more efficient and effective. Numerical results to demonstrate these improvements are presented.

1 Introduction and motivation

Transistor- and wire-sizing is essential for the design of optimal high-performance digital circuits. Automatic circuit optimization also leads to enhanced designer productivity. Static optimization simultaneously optimizes all paths through the logic, since it is based on static timing analysis. TILOS [1] is a well-known example of a static circuit optimizer.

In [2, 3], a method of formulating static circuit optimization as a standard nonlinear optimization problem is described. In [3], time-domain simulation and sensitivity analysis of each circuit block is used to time and optimize the circuit on this basis. Formal nonlinear optimization is employed to obtain an optimal solution. The disadvantage of such a formulation is that the number of variables and constraints can be very large, causing the solution to be inefficient or even intractable. Further, the presence of inherent redundancy and degeneracy in the problem statement causes numerical difficulties for the nonlinear optimizer.

In this paper, we describe a simple timing graph manipulation algorithm that re-states static circuit optimization problems in an equivalent but compact form with reduced degeneracy and redundancy. There is no loss of accuracy or generality. However, the transfor-

mation renders the problem amenable to more efficient solution. The method is generally applicable to static circuit optimizers that use an “arrival-time-based” formulation of the problem.

The algorithm has been implemented in a static circuit optimization tool called EinsTuner [3], which tunes digital circuits composed of combinational parameterized cells. On average, an 88.0% reduction in arrival time variables, a 25.3% reduction in the total number of variables, a 39.3% reduction in timing constraints and an 18.7% reduction in total constraints were realized by the application of this algorithm. After optimization, a consistent reduction of about 40% in the number of degenerate constraints is seen at the solution. As a consequence, in many cases the number of conjugate gradient iterations and the CPU time are significantly reduced (by more than 30% for the former, which translates to about a 20% reduction in CPU time) while being accompanied by a better solution.

The outline of the rest of the paper is as follows. Section 2 describes the previous formulation of the static circuit optimization problem. The basic “pruning” operation to reduce size, degeneracy and redundancy is described in Section 3, and the algorithm is demonstrated by means of an example in Section 4. Section 5 discusses some implementation details and numerical results are presented in Section 6. Conclusions and future work are outlined in Section 7.

2 Static circuit optimization

An understanding of the previous formulation of static circuit optimization and its advantages and disadvantages is essential to understanding the benefits of the timing graph manipulation. This section provides that background information first by means of an example and then in a more formal manner.

2.1 Formulation by example

Consider the network of Figure 1, consisting of three simple gates G_1 , G_2 and G_3 . Let \mathbf{w} be the n -vector of transistor widths to be optimized. Separate falling and rising arrival times, slews, slew dependencies and the delay through wires have been ignored here for simplicity. Assume that we wish to minimize the worst arrival time

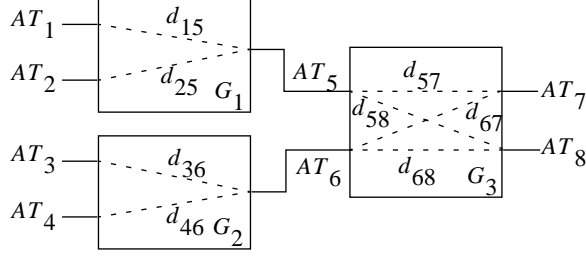


Figure 1: Simple example to illustrate the problem formulation.

at the primary outputs subject to an area constraint and simple bounds on transistor widths. The problem is formulated as follows.

$$\begin{aligned}
\min \quad & z \\
\text{s.t.} \quad & z \geq AT_7 \\
& z \geq AT_8 \\
& AT_7 \geq AT_5 + d_{57}(\mathbf{w}) \\
& AT_7 \geq AT_6 + d_{67}(\mathbf{w}) \\
& AT_8 \geq AT_5 + d_{58}(\mathbf{w}) \\
& AT_8 \geq AT_6 + d_{68}(\mathbf{w}) \\
& AT_5 \geq AT_1 + d_{15}(\mathbf{w}) \\
& AT_5 \geq AT_2 + d_{25}(\mathbf{w}) \\
& AT_6 \geq AT_3 + d_{36}(\mathbf{w}) \\
& AT_6 \geq AT_4 + d_{46}(\mathbf{w}) \\
& \sum_i k_i w_i \leq A \\
& L_i \leq w_i \leq U_i.
\end{aligned} \tag{1}$$

In problem (1), the AT variables are the worst-case arrival times at each node of the circuit, the d_{ij} functions are the delays of *propagate segments* from each input pin to the output pin of each gate, A is an area target, and L_i and U_i are the lower and upper bounds on transistor widths (known as simple bounds). Area is modeled by a weighted sum of transistor widths, and z is an auxiliary variable introduced to solve the problem. The arrival times at non-primary-inputs are variables of the optimization problem, just like the transistor widths. Input arrival times are provided in the form of timing assertions. The optimization is carried out in the space $\mathbf{w} \in \mathbb{R}^n$; $z \in \mathbb{R}$; $AT_i, i = 5, 6, 7, 8 \in \mathbb{R}^4$.

Problem (1) expresses a standard smooth nonlinear optimization problem without discontinuities in the first derivatives (provided the d_{ij} functions are continuously differentiable in \mathbf{w}). The problem can equally be formulated as a minimization of area subject to a timing requirement, or the minimization of a weighted sum of critical path delay (or negative slack) and area.

2.2 Problem formulation

While the previous subsection illustrates the basic idea of the formulation by means of a simple example, the detailed formulation is spelled out in this subsection. Each node of the circuit has four variables associated with it. For node i , the four variables are the rising arrival time AT_i^r , the falling arrival time AT_i^f , the rising slew S_i^r and the falling slew S_i^f . Next, we have the auxiliary variable

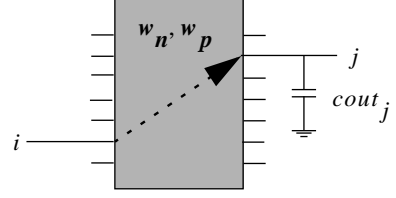


Figure 2: Generic channel-connected component.

z . Assume that we are dealing with parameterized gates in which each CCC (channel-connected component or set of source-drain connected transistors) has two design variables, w_n and w_p , denoting parameters that control the width of all the NFETs and all the PFETs in the CCC, respectively. The formulation can easily be generalized to a full-custom situation.

Figure 2 shows a generic multi-input multi-output CCC. For the propagate segment from pin i to pin j , the constraints are shown below, assuming that the segment is an inverting segment. The entire network is traversed and constraints such as the ones listed below are gathered for every propagate segment.

$$\begin{aligned}
AT_j^r &\geq AT_i^f + d_{ij}^r(w_n, w_p, cout_j, S_i^f) \\
AT_j^f &\geq AT_i^r + d_{ij}^f(w_n, w_p, cout_j, S_i^r) \\
S_j^r &\geq s_{ij}^r(w_n, w_p, cout_j, S_i^f) \\
S_j^f &\geq s_{ij}^f(w_n, w_p, cout_j, S_i^r).
\end{aligned} \tag{2}$$

Note that a superscript of r implies a rising signal, delay or slew, while a superscript of f refers to the corresponding falling quantities. The nonlinear delay functions are denoted by d_{ij} , and the s_{ij} terms represent the nonlinear slew functions. The fanout capacitance at pin j is $cout_j$, which is a function of the sizes of the fanouts of pin j augmented by any wire capacitances on that net. The gate itself is modeled at the transistor level. In keeping with the conservative nature of late-mode static timing analyzers, a conservative slew propagation method is employed and the worst slew is propagated even if it does not correspond to the last-arriving signal.

Additional constraints and/or simple bounds can be added to constrain slews, primary input loading, area, β ratios (w_p/w_n), and so on. Most commonly, we minimize an auxiliary variable z which in turn is constrained to exceed the arrival times at all the primary outputs. Further, additional timing constraints can be added to extend the formulation to dynamic logic and sequential elements. In all cases, the timing and slew constraints for the combinational part of the network have the general form shown in (2) above and the final statement is in the form of a standard nonlinear optimization problem.

2.3 Advantages and disadvantages

The problem formulation described above is general and comprehensive. A solution to the problem ensures that the worst of all path delays is minimized. Further, the framework is flexible and amenable to the statement of

various types of additional constraints and variations on the optimization problem.

This representation suffers from some disadvantages, the main one being the size of the resulting optimization problem, both in terms of number of variables and number of constraints. Moderate-sized circuits can lead to very large optimization problems, thus stressing the limits of capacity and performance of nonlinear optimization packages. For example, a circuit with 1,529 gates generates 9,245 variables, 5,164 timing constraints, 5,164 slew constraints and a total of 10,329 constraints!

We note that at the solution, by definition, the auxiliary variable z is at its minimum value, while all the constraints are feasible. In addition, the constraints along all critical paths will be tight, and the arrival times along these paths will be correct. Timing and slew constraints off the critical path may not be tight and these variables can take one of several equally correct values at the solution. Thus the problem formulation has a high degree of inherent redundancy and degeneracy, as noted in [4].

Degeneracy implies a zero Lagrange multiplier associated with an active constraint. A constraint is redundant if merely removing it does not change the problem. All active redundant constraints give rise to degenerate problems. However, there are degenerate problems that have no redundancies. The optimizer finds a solution by determining the active constraints, which is achieved by estimating the corresponding multipliers, and in particular, determining their sign. In the case of redundant or degenerate constraints, the multipliers and/or local solutions are not uniquely defined and consequently the optimization algorithm has difficulty making the correct choices. The phenomenon occurs not only in the neighborhood of the solution but also at prospective stationary points along the way. The non-uniqueness of the Lagrange multipliers causes the optimizer difficulty both in identifying the “active set” (set of tight constraints) and the correct values of the multipliers. Indeed, degeneracy and redundancy reduce the effectiveness and efficiency of the optimizer. Numerical noise, which is inevitable in any simulation-based approach, exacerbates this situation. Thus eliminating degeneracy in the presence of numerical noise has added benefit.

3 Pruning

This section presents the pruning algorithm that alleviates the disadvantages of this type of problem formulation.

3.1 A simple example

Consider a fragment of a circuit shown in Fig. 3. The timing constraints in which the arrival time AT_3 appears are

$$\begin{aligned} AT_3 &\geq AT_1 + d_{13} \\ AT_3 &\geq AT_2 + d_{23} \\ AT_4 &\geq AT_3 + d_{34} \\ AT_5 &\geq AT_3 + d_{35}. \end{aligned} \quad (3)$$

Slews and separate rising/falling arrival times have been omitted for simplicity. An equivalent set of constraints is

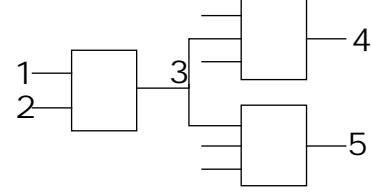


Figure 3: Fragment of a circuit.

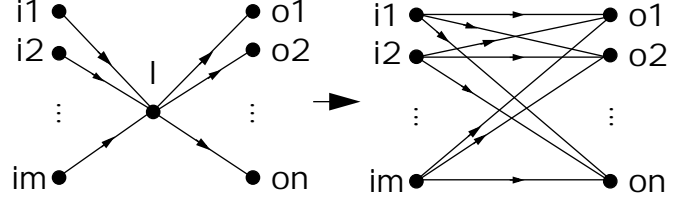


Figure 4: Basic graph manipulation.

$$\begin{aligned} AT_4 &\geq AT_1 + d_{13} + d_{34} \\ AT_4 &\geq AT_2 + d_{23} + d_{34} \\ AT_5 &\geq AT_1 + d_{13} + d_{35} \\ AT_5 &\geq AT_2 + d_{23} + d_{35}. \end{aligned} \quad (4)$$

It can be seen that by choosing

$$AT_3 = \max(AT_1 + d_{13}, AT_2 + d_{23}), \quad (5)$$

the set of constraints in (4) is equivalent to the set of constraints in (3). By this simple manipulation, we went from a situation in which we had 5 arrival time variables and 4 constraints to a new situation with 4 arrival time variables and 4 constraints – a reduction in the dimensionality of the problem. Of course, each constraint now has extra terms. In fact, the summations of block delays can be thought of as path delays. However, the number of CCC simulations is unchanged, and having additional nonlinear terms in a constraint is not harmful to the optimizer we use since it exploits *group partial separability* ([5], page 109) and the overall number of distinct terms does not increase. We observe that pruning in this way is possible only because the arrival times do not appear nonlinearly in the original constraints.

Perhaps more importantly, we have eliminated the chance of AT_3 being degenerate at the solution and reduced the probability of the associated constraints being degenerate, thus relieving to some extent the burden of the nonlinear optimization package.

3.2 Formal description of pruning

Consider a segment of a graph shown on the left side of Fig. 4. The node l will be pruned. This node has m fanins (m edges are incident on this node), and n fanouts (n edges originate at this node). The timing constraints for this graph segment are shown below.

$$\begin{aligned} AT_l &\geq AT_{i_j} + d_{i_j, l} \quad \forall j \in 1, 2, \dots, m \\ AT_{o_k} &\geq AT_l + d_{l, o_k} \quad \forall k \in 1, 2, \dots, n. \end{aligned} \quad (6)$$

We now consider eliminating or pruning AT_l from the above equations, to obtain

Table 1: “Gain” from pruning operations.

		# fanins (m)			
		1	2	3	4
# fan-outs (n)	1	4	4	4	4
	2	4	2	0	-2
	3	4	0	-4	-8
	4	4	-2	-8	-14

$$AT_{ok} \geq AT_{ij} + d_{ij,l} + d_{l,ok} \quad \forall j \in 1, 2, \dots, m, \quad \forall k \in 1, 2, \dots, n. \quad (7)$$

The constraints in the above equation are shown graphically on the right side of Fig. 4. It can be seen that by choosing

$$AT_i = \max(AT_{ij} + d_{ij,i}) \quad \forall i \in 1, 2, \dots, m \quad (8)$$

the “pruned” set of constraints is equivalent to the original set of constraints. We note that slews cannot be pruned in a straightforward manner since they occur nonlinearly in the original constraints.

The above pruning procedure replaces $2(m+n)$ constraints and $2(m+n+1)$ arrival time variables by $2(mn)$ constraints and $2(m+n)$ arrival times. The factor of 2 takes into account separate variables for rising and falling arrival times, and the concomitant timing constraints. We define the “gain” of a pruning operation to be the reduction in problem size, i.e., the reduction in the total number of variables and constraints. Table 1 shows the gain by applying a pruning operation for various values of m and n . For some values of m and n , the gain is 4, and therefore the pruning procedure is clearly advantageous. Even when the gain is 0, there may be an advantage to pruning, since it reduces the number of arrival time variables by two, leading to possibly reduced redundancy and/or degeneracy. When the gain is negative, however, it is unlikely that pruning will help. *We observe that most practical networks have graph structures that lend themselves to tremendous gains by pruning.* A weighted sum of the number of constraints and variables can be used to tailor the gain computation to a particular optimizer. We observe that pruning is similar in some respects to methods used to produce compact delay model abstractions.

In the modified problem formulation after pruning, slews can be handled in one of two ways. First, all slew variables can be retained and slew constraints left untouched. In this case, each CCC is simulated as before and slew propagation is unchanged. No reduction in the number of slew variables is obtained. The second method is to treat each edge of the pruned graph as a single “super” propagate segment. The simulator times each of the blocks on this segment using an input slew that is propagated exclusively along this segment and returns to the optimizer the total delay and slew of the super segment and the gradients thereof. Each CCC may

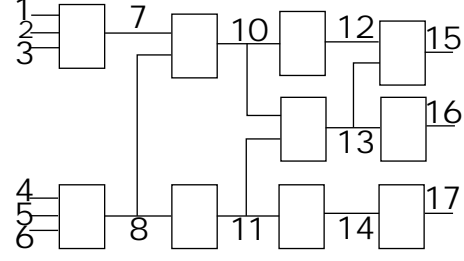


Figure 5: Sample circuit.

have to be simulated multiple times, but the pessimism in slew propagation is reduced. Even better, the entire path can be fed to the transient simulator so that actual analog waveforms are propagated from one stage to the next within the super segment. In this method, as many slew variables are pruned as arrival time variables. The first method (retaining the original slew variables) is currently implemented in the software described in this paper.

4 Example of pruning procedure

In this section, the graph representing a sample combinational circuit is pruned to demonstrate the benefits of pruning. Fig. 5 shows a sample circuit and Fig. 6 its timing graph, containing 26 edges and 16 nodes. Source and sink nodes are introduced with edges to all the primary inputs and outputs, respectively, and are not included in the node count. The timing constraints of this circuit prior to any pruning procedure are shown below. It is assumed below that we are trying to minimize the cycle time of the circuit, represented by the variable z . *RAT* implies “required arrival time” and *AT* “arrival time.”

$$\begin{aligned}
 \min \quad & z \\
 \text{s.t. } z & \geq RAT_{15} + d_{12,15} + AT_{12} \\
 \text{s.t. } z & \geq RAT_{15} + d_{13,15} + AT_{13} \\
 \text{s.t. } z & \geq RAT_{16} + d_{13,16} + AT_{13} \\
 \text{s.t. } z & \geq RAT_{17} + d_{14,17} + AT_{14} \\
 \text{s.t. } AT_{12} & \geq d_{10,12} + AT_{10} \\
 \text{s.t. } AT_{13} & \geq d_{10,13} + AT_{10} \\
 \text{s.t. } AT_{13} & \geq d_{11,13} + AT_{11} \\
 \text{s.t. } AT_{14} & \geq d_{11,14} + AT_{11} \\
 \text{s.t. } AT_{10} & \geq d_{7,10} + AT_7 \\
 \text{s.t. } AT_{10} & \geq d_{8,10} + AT_8 \\
 \text{s.t. } AT_{11} & \geq d_{8,11} + AT_8 \\
 \text{s.t. } AT_7 & \geq d_{1,7} + AT_1 \\
 \text{s.t. } AT_7 & \geq d_{2,7} + AT_2 \\
 \text{s.t. } AT_7 & \geq d_{3,7} + AT_3 \\
 \text{s.t. } AT_8 & \geq d_{4,8} + AT_4 \\
 \text{s.t. } AT_8 & \geq d_{5,8} + AT_5 \\
 \text{s.t. } AT_8 & \geq d_{6,8} + AT_6.
 \end{aligned} \quad (9)$$

Nodes 1, 2, 3, 4, 5, 6, 14, 15, 16 and 17 are observed to yield a gain of 4, and hence are pruned to obtain the graph in Fig. 7. The edges of the pruned graph are annotated with the pruned arrival times and this annotation will be required to generate the timing constraints later. Next, nodes 7, 11 and 12, with gains of 4, are pruned

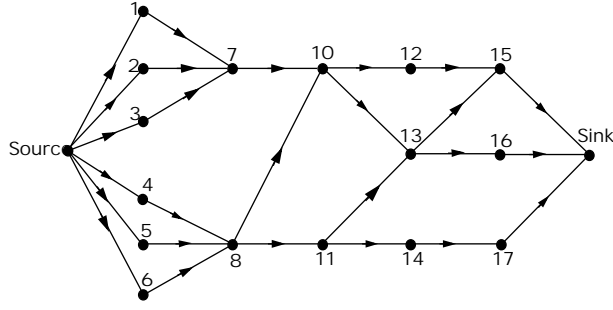


Figure 6: Propagate segment graph of sample circuit.

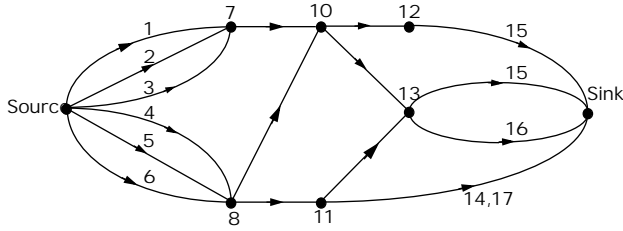


Figure 7: Timing graph of sample circuit after pruning nodes 1, 2, 3, 4, 5, 6, 14, 15, 16 and 17.

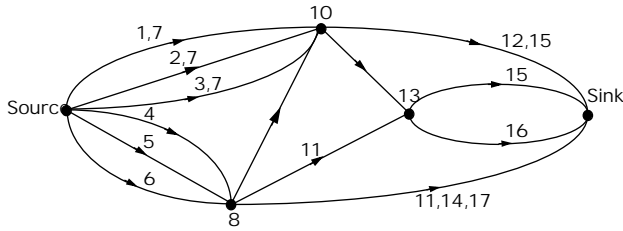


Figure 8: Timing graph of sample circuit after pruning nodes 7, 11 and 12.

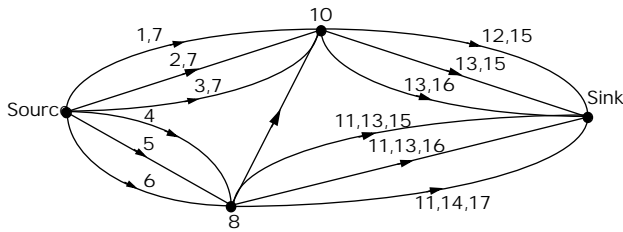


Figure 9: Final pruned timing graph of sample circuit after pruning node 13.

to obtain the graph in Fig. 8. Node 13 with a gain of 2 is then pruned to obtain the final graph in Fig. 9. The number of nodes in the graph has been reduced from 16 to 2 (an 8x reduction in this example) and the number of edges from 26 to 13 (a 2x reduction in this example). The timing constraints corresponding to the final pruned graph are shown below. Obviously, this more compact formulation of the problem is easier and more efficient to solve. The pruning procedure takes very little computer time, whereas the savings in computer time during the optimization can be very large.

$$\begin{aligned}
 \min \quad & z \\
 \text{s.t. } z \quad & \geq RAT_{15} + d_{12,15} + d_{10,12} + AT_{10} \\
 \text{s.t. } z \quad & \geq RAT_{15} + d_{13,15} + d_{10,13} + AT_{10} \\
 \text{s.t. } z \quad & \geq RAT_{16} + d_{13,16} + d_{10,13} + AT_{10} \\
 \text{s.t. } z \quad & \geq RAT_{15} + d_{13,15} + d_{11,13} + d_{8,11} + AT_8 \\
 \text{s.t. } z \quad & \geq RAT_{16} + d_{13,16} + d_{11,13} + d_{8,11} + AT_8 \\
 \text{s.t. } z \quad & \geq RAT_{17} + d_{14,17} + d_{11,14} + d_{8,11} + AT_8 \\
 \text{s.t. } AT_{10} \quad & \geq d_{7,10} + d_{1,7} + AT_1 \\
 \text{s.t. } AT_{10} \quad & \geq d_{7,10} + d_{2,7} + AT_2 \\
 \text{s.t. } AT_{10} \quad & \geq d_{7,10} + d_{3,7} + AT_3 \\
 \text{s.t. } AT_{10} \quad & \geq d_{8,10} + AT_8 \\
 \text{s.t. } AT_8 \quad & \geq d_{4,8} + AT_4 \\
 \text{s.t. } AT_8 \quad & \geq d_{5,8} + AT_5 \\
 \text{s.t. } AT_8 \quad & \geq d_{6,8} + AT_6.
 \end{aligned}$$

(10)

We emphasize that no approximations are being made. The representation before and after pruning are completely equivalent.

4.1 Observations

The number of fanins and fanouts of a node increase monotonically during the pruning procedure. Therefore, for a given pruning criterion (gain threshold), if a node is unsuitable for pruning, it will never be pruned under that same criterion. Hence a single pass through the graph is sufficient to prune nodes based on a given criterion, e.g., to prune nodes with a gain of 4.

The order of pruning is significant. Pruning a node may cause the gain of a neighboring node to decrease, moving it below the pruning threshold. Hence it is desirable to first prune nodes that yield the most gain. While our implementation does not seek to solve the pruning problem optimally, the greedy implementation described in the following section performs well on a wide variety of circuits.

When an inequality constraint is removed, the corresponding slack variable introduced (internal to the optimizer) to convert it to an equality is also eliminated. The computation of gain can easily be modified to take into account the reduction in slack variables as a result of pruning.

In timing graphs that include wire segments, most arrival times along wires have just one fanin, so extensive pruning is likely to occur. In sequential circuits, if certain arrival times appear in additional constraints such as set-up constraints, it may not always be possible to prune those timing points.

Graph theory literature mentions graph operations such as deletion of nodes and addition of edges in several contexts. For example, one may delete edges via a *contraction* operator. Similarly, there is a *closure* operation for adding edges to a graph (see, for example, [6]). In graph theoretic terms, our pruning could be described as squaring the relevant subgraph determined by the node to be deleted and its adjacent nodes, and subsequently deleting the internal node and the original edges connected to it. There is a rich collection of operations on abstract graphs but no examples, to the best of our knowledge, of our particular form of node deletion. In any case, the significance of our pruning algorithm is in the application to particular digraphs that represent a formulation of static circuit optimization problems.

While our description was conveniently stated in terms of timing graphs, pruning of arrival times can of course be achieved purely algebraically. One could also consider pruning sets of adjacent nodes at once; all paths through the sub-graph being pruned are listed as edges of the pruned graph.

5 Implementation

The pruning algorithm was implemented in a static circuit optimization tool called EinsTuner [3]. The circuit is first read and the corresponding timing graph constructed. All nodes of gain 4 are pruned in an initial pass of the pruning algorithm. Nodes with a gain of 2 and 0 are pruned in subsequent sequential passes. Thus “greedy 3-pass pruning” is applied to reduce the size of the graph as much as possible. Then the graph is translated into a nonlinear problem formulation and fed to the large-scale, general-purpose optimizer LANCELOT [5, 7, 8]. When the optimizer asks for delay and slew values, the fast circuit simulator SPECS [9, 10] is invoked to compute the required delays and slews. SPECS employs efficient time-domain adjoint sensitivity computation to also return the gradients of each slew and delay measurement with respect to transistor widths, input slews and loading capacitance [11, 12, 13]. LANCELOT digests all the function and gradient information to come up with the solution vector at the next iteration. The optimization iterations continue until convergence is obtained. Special stopping criteria and techniques to deal with numerical noise have been implemented in this optimization framework. “Two-step updating” [14] of variables that occur linearly is employed to enhance optimization efficiency.

EinsTuner allows several options to specify additional constraints, e.g., internal slew limits, output slew limits, effective β -ratio constraints, input loading constraints and an area constraint. Grouping and tunability information are specified in a control file. Thus certain instances or gate types can be declared to be untunable, and certain other instances “grouped” so that they can share a layout after the tuning.

At the end of the optimization, optimal transistor widths are back-annotated and a final timing run conducted to determine the correct arrival times and

slews at every point in the network. Infeasibilities on any of the inherent timing and slew constraints or the user-specified additional constraints are checked and reported.

The actual pruning is implemented by repeatedly checking each unpruned node against a gain criterion. If the criterion is met, the node is marked as pruned. In the next pass, the gain criterion is lowered and the process repeated. At each pass, care must be taken to compute the correct number of fanins and fanouts of each node taking into account previous pruning decisions. When the pruning is finished, a simple depth-first traversal of the resulting annotated pruned graph yields the list of timing constraints.

6 Numerical results

EinsTuner was run on 21 benchmark circuits ranging in size from 3 to 1,529 parameterized gates (6 to 5,164 transistors), with and without the pruning algorithm. Of the problems, 9 were macros from high-performance S/390 and PowerPC microprocessors (labeled “s390” and “ppc” in the results), and the other 12 were ISCAS-85 or internal benchmarks. First, we report on problem size. The fourth major column of Table 2 shows the number of arrival time variables before and after pruning, leading to an average 88.0% reduction in the number of arrival time variables. The next major column reports the reduction in the total number of variables, averaging 25.3%. Then the reduction in the number of timing constraints and total number of constraints is shown. Clearly, tremendous compaction of the optimization problems is achieved by the pruning procedure.

At the end of the optimization, LANCELOT reports on whether each variable or constraint was degenerate at the solution. The second and third major columns of Table 3 show the reduction in degeneracy in all the constraints and in just the timing constraints, respectively. Thus in addition to significant reduction in the size of the problem, the pruning procedure demonstrably reduces constraint degeneracy in the problem statement, as expected. Pruning achieves a marked gain on some of the larger problems.

The situation with degenerate variables is more complex. With the exception of s390-3 (where the number was reduced from 1 to 0) there were no degenerate timing variables or slews. However, many transistor widths off the critical path were at their lower bounds at the solution with very small corresponding Lagrange multipliers. Numerical noise, the specialized stopping criteria that we employ and the smallness of the multipliers make it difficult to accurately count transistor width degeneracies. In an attempt to more correctly account for degenerate variables, we examined three problems carefully with more stringent stopping criteria. In all cases there was a decrease in the number of degenerate transistor width variables as a result of pruning (in c432, for example, from 376 to 370 and in c8, from 292 to 252).

Table 4 shows the results of actual circuit optimization. The tests were run on a pool of IBM Risc/System

Table 2: Reduction in problem size due to pruning.

Name	Gates	Trans.	Arrival time variables			Total variables			Timing constraints			Total constraints				
			w/o	w/	Redn.	w/o	w/	Redn.	w/o	w/	Redn.	w/o	w/	Redn.		
inv3	3	6	5	1	80%	19	15	21%	6	2	66%	13	9	30%		
c17	7	28	11	1	90%	49	39	20%	28	22	21%	57	51	10%		
a3_3	9	34	13	5	61%	55	47	14%	34	28	17%	69	63	8%		
andy_graph	10	34	15	5	66%	67	57	14%	34	26	23%	69	61	11%		
full_ladder	14	46	25	3	88%	87	65	25%	46	24	47%	93	71	23%		
s390-1	22	72	43	1	97%	147	105	28%	72	32	55%	145	105	27%		
s390-2	24	102	45	3	93%	153	111	27%	102	60	41%	259	217	16%		
s390-3	41	154	59	5	91%	241	187	22%	154	104	32%	318	268	15%		
ppc-1	155	824	309	1	99%	899	591	34%	824	516	37%	1796	1488	17%		
s390-4	175	882	223	21	90%	935	733	21%	882	690	21%	1765	1573	10%		
c8	180	584	327	13	96%	1103	789	28%	584	280	52%	1169	865	26%		
ppc-2	218	880	361	51	85%	1337	1027	23%	880	626	28%	2249	1995	11%		
c432	299	960	591	23	96%	1859	1291	30%	960	440	54%	1921	1401	27%		
ppc-3	408	1554	689	85	87%	2449	1845	24%	1554	1072	31%	3109	2627	15%		
s390-5	430	1400	859	45	94%	2803	1989	29%	1400	640	54%	3773	3013	20%		
c880	481	1582	915	107	88%	2959	2151	27%	1582	838	47%	3165	2421	23%		
ppc-4	628	2726	1207	125	89%	3919	2837	27%	2726	1724	36%	6809	5807	14%		
c1355	665	2180	1267	177	86%	4009	2919	27%	2180	1206	44%	4361	3387	22%		
c499	667	2216	1271	183	85%	4021	2933	27%	2216	1274	42%	4433	3491	21%		
c2670	837	2796	1603	251	84%	5261	3909	25%	2796	1484	46%	5593	4281	23%		
c3540	1529	5164	3029	373	87%	9245	6589	28%	5164	2748	46%	10329	7913	23%		
Average not counting inv3					88.0%				25.3%				39.3%			18.7%

Table 3: Reduction in degeneracy due to pruning.

Name	Degen. const.		Degen. timing const.	
	w/o	w/	w/o	w/
inv3	0	0	0	0
c17	0	0	0	0
a3_3	1	2	1	1
andy_graph	3	0	1	0
full_ladder	3	1	1	0
s390-1	9	11	0	0
s390-2	9	9	1	0
s390-3	28	19	10	0
ppc-1	286	261	62	0
s390-4	105	90	24	5
c8	96	47	48	1
ppc-2	242	117	116	4
c432	238	116	122	1
ppc-3	171	136	89	15
s390-5	363	330	62	9
c880	512	228	226	8
ppc-4	763	373	409	14
c1355	451	311	138	5
c499	519	276	258	15
c2670	970	492	577	66
c3540	1134	538	536	24

6000 workstations. All runs of a particular benchmark were performed on the same machine, so as to be able to compare CPU times. The 3-pass pruning algorithm takes a fraction of a second of CPU time even on the largest benchmarks in this table. The starting point on the ISCAS benchmarks was obtained by a heuristic gain-based sizing algorithm. In all other cases, the actual sets of transistor sizes provided by the designers were employed as start points. The second major column shows the starting critical path delay of the circuit. The final tuned result with and without pruning is shown in the third major column along with the percentage improvement of the pruned result over the original delay. The final projected gradient, number of optimization iterations, total number of conjugate gradient iterations in LANCELOT and total CPU time with and without

pruning are shown in subsequent columns.

Excluding inv3, the average percentage decrease in conjugate gradient iterations was 32.6% with a corresponding figure of 13.4% for the CPU time. The average improvement in the cycle time was 13.8%. Although the results for s390-3 look poor, in fact the original starting point had significant slew and input capacitance infeasibilities, while the final point was feasible. For 4 of the problems, there was an average *increase* of 45.7% in the CPU time, but the solution found was significantly better. Indeed, in all cases one can make an argument that the pruned results are superior, either because they were obtained more rapidly, or the solution or final projected gradient was smaller. However, in the case of c17 and s390-4 the results could be considered unsatisfactory to some extent. For c17 the solution was obtained with a reduction in CPU time of 70.1% but was 15.9% worse than the non-pruning result. However, with different optimizer settings, we obtained a marginally better solution than without pruning, with a slight reduction in CPU time. With s390-4, the slightly worse solution was at a saving in CPU time of 17.2%. With the same different parameter settings as mentioned above, the solution was essentially the same with and without pruning with the same CPU time.

7 Conclusions and future work

Previous static circuit optimization formulations lead to very large nonlinear optimization problems with inherent redundancy and degeneracy in them. In this paper, a method of pruning the timing graph was described to reduce the size of the resultant nonlinear optimization problem. Additionally, the numerical qualities of the problem statement were improved in terms of reduced

Table 4: Numerical results with and without pruning.

Name	Start delay (ps)	Tuned delay (ps)			Proj. grad.		# its.		C.G. its.		CPU (s)	
		w/o	w/	Redn.	w/o	w/	w/o	w/	w/o	w/	w/o	w/
inv3	123.3	78.3	86.7	29.7%	0.04	0.04	31	14	381	116	7.8	5.7
c17	200.1	123.8	143.5	28.3%	0.00	0.05	78	19	11495	1415	38.3	11.4
a3_3	320.5	203.7	204.5	36.2%	0.30	0.03	44	45	8708	9451	65.6	68.3
andy_graph	299.7	262.7	263.4	12.1%	0.03	0.02	25	27	1353	1153	34.9	37.0
full_adder	310.2	229.5	229.6	26.0%	0.03	0.03	72	65	15308	10175	113.5	99.6
s390-1	275.0	259.3	262.1	4.7%	0.03	0.02	64	20	9476	710	157.3	52.2
s390-2	380.4	339.9	350.2	7.9%	0.01	0.07	84	69	44727	17620	781.4	600.7
s390-3	1568	1579	1570	-0.8%	0.36	0.03	19	23	5786	2496	63.3	68.5
ppc-1	584.2	526.4	530.3	9.2%	0.07	0.09	30	16	28981	2761	2708	1174
s390-4	306.2	257.0	259.4	15.3%	0.05	0.06	27	26	19049	8335	1689	1399
c8	712.3	591.4	560.6	21.3%	0.17	0.08	27	60	16218	33998	727.0	1505
ppc-2	627.6	563.3	553.8	11.8%	0.04	0.07	46	56	101987	140020	5187	6885
c432	1431	1171	1160	18.9%	0.13	0.26	64	38	66682	31687	3345	2142
ppc-3	624.0	606.5	593.2	4.9%	0.03	0.04	40	35	36649	19561	4527	3664
s390-5	943.5	623.4	609.9	35.4%	0.01	0.03	228	144	281435	190719	27125	20825
c880	1597	1355	1336	16.4%	0.59	0.59	97	38	126288	30541	10285	3629
ppc-4	1277	1091	1146	10.2%	0.08	0.08	159	68	7821824	2168505	284445	115390
c1355	1218	1081	1103	9.5%	0.05	0.05	48	48	263368	136393	21446	13375
c499	1242	1101	1084	12.8%	0.05	0.04	49	40	210909	139526	4728	3654
c2670	1082	976.1	959.2	11.4%	0.08	0.04	55	53	160637	105839	20114	15788
c3540	2269	2132	1997	12.0%	1.10	0.06	67	107	207914	194069	18520	24922

degeneracy and redundancy. Experimental evidence was presented to demonstrate reduced problem size, better numerical qualities and better numerical results. The efficiency and/or effectiveness of circuit optimization was shown to improve as a result of the pruning of the timing graph.

Several avenues of future work suggest themselves. While the three-pass greedy pruning performs well on a wide range of circuit, it may not obtain the optimal pruning result. One could theoretically consider listing all possible paths from the primary inputs to the primary outputs and then re-introducing a minimal set of timing points to minimize the sum of the nodes and edges in the graph. In the implementation described in this paper, slew variables were not pruned. However, this paper describes how slews could be pruned, which would be a good future topic of investigation. Finally, static optimization could be equally well applied to early mode and simultaneous early and late mode optimization.

8 Acknowledgments

The authors would like to gratefully acknowledge the entire EinsTuner team, especially E. K. Cho, I. M. Elfadel, G. T. Kaminsky, D. D. Ling, W. W. Molzen, Jr., P. R. O'Brien and P. N. Strenski for useful discussions, support and help with software. We would also like to thank I. M. Elfadel and R. A. Haring for useful comments and suggestions on the manuscript.

References

- [1] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," *IEEE International Conference on Computer-Aided Design*, pp. 326-328, November 1985.
- [2] C.-P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian Relaxation," *IEEE International Conference on Computer-Aided Design*, pp. 617-624, November 1998.
- [3] A. R. Conn, I. M. Elfadel, W. W. Molzen, Jr., P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan, "Gradient-based optimization of custom circuits using a static-timing formulation," *Proc. 1999 Design Automation Conference*, pp. 452-459, June 1999.
- [4] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance driven placement algorithm for small cell ICs," *IEEE International Conference on Computer-Aided Design*, pp. 48-51, November 1991.
- [5] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Verlag, 1992.
- [6] J. A. Bondy and U. S. R. Murty, *Graph theory with applications*. North-Holland, 1976.
- [7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 433-460, 1988. See also same journal, pp. 764-767, volume 26, 1989.
- [8] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545-572, 1991.
- [9] C. Visweswariah and R. A. Rohrer, "Piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 861-870, July 1991.
- [10] C. Visweswariah and J. A. Wehbeh, "Incremental event-driven simulation of digital FET circuits," *Proc. 1993 Design Automation Conference*, pp. 737-741, June 1993.
- [11] P. Feldmann, T. V. Nguyen, S. W. Director, and R. A. Rohrer, "Sensitivity computation in piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 171-183, February 1991.
- [12] A. R. Conn, R. A. Haring, C. Visweswariah, and C. W. Wu, "Circuit optimization via adjoint Lagrangians," *IEEE International Conference on Computer-Aided Design*, pp. 281-288, November 1997.
- [13] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah, and C. W. Wu, "JiffyTune: circuit optimization using time-domain sensitivities," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 17, pp. 1292-1309, December 1998.
- [14] A. R. Conn, L. N. Vicente, and C. Visweswariah, "Two-step algorithms for nonlinear optimization with structured applications," Research Report RC 21198(94689), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, June 1998. Accepted for publication in *SIAM Journal on Optimization*.