Eyecharts: Constructive Benchmarking of Gate Sizing Heuristics

Amarnath Kasibhatla
amar@ee.ucla.edu
University of California Los Angeles
Master's Project Report, Spring 2010
Advisor: Prof. Puneet Gupta

***Abstract*—This report presents extensions to the dynamic programming-based framework proposed in [1] for creating optimally sized benchmark circuits called *eyecharts*. Discrete gate sizing is one of the most commonly used, flexible, and powerful techniques for digital circuit optimization. The underlying problem has been proven to be NP-hard [2]. Several (suboptimal) gate sizing heuristics have been proposed over the past two decades, but research has suffered from the lack of any systematic way of assessing the quality of the proposed algorithms. We develop a method to generate benchmark circuits (called *eyecharts*) of arbitrary size along with a method to compute their optimal solutions using dynamic programming. We evaluate the suboptimalities of some popular gate sizing algorithms. Eyecharts help diagnose the weaknesses of existing gate sizing algorithms, enable systematic and quantitative comparison of sizing algorithms, and catalyze further gate sizing research. Our results show that common sizing methods (including commercial tools) can be suboptimal by as much as 54% (V_t-assignment), 46% (gate sizing) and 49% (gate-length biasing) for realistic libraries and circuit topologies.**

I. Introduction

The *sizing problem* in digital VLSI design seeks to tune the circuit parameters of supply voltage, threshold voltage, gate-length and gate-width to optimize a tradeoff of speed, area and power. The sizing problem arises at all stages of the RTL-to-GDS implementation flow, and even beyond (e.g. [3]). The classical problem of discrete gate sizing is to assign a size (from a pre-characterized cell library) to each gate in a combinational logic block, such that the block's total power is minimized, subject to a maximum delay constraint. Finding the optimal gate sizing solution for a given digital logic circuit can be NP-hard [2].

Fishburn and Dunlop proposed a fast greedy method, TILOS [4], to minimize area while meeting delay constraints. Chan [5] gives a pseudo-polynomial time slack-computation algorithm and a backtracking algorithm for gate sizing. Previous methods have also used mathematical programming techniques to do gate sizing: linear programming (LP) [6]–[9], Lagrangian relaxation [10], [11], and convex optimization [12]–[14]. Other methods include sensitivity-based approaches [3], [15]–[17], dynamic programming (DP) [18], [19] and heuristics guided by continuous programming [20]. Coudert et al. [16] give a good comparison of the gate sizing algorithms proposed during the early 1990s.

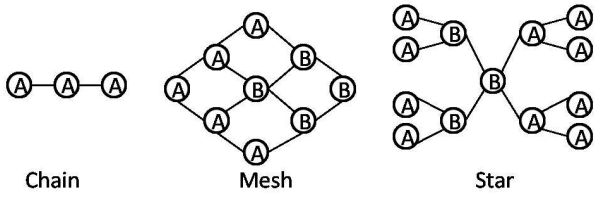


Fig. 1. Basic eyechart topologies.

None of the previous methods in the literature (except in [21], [22]) quantify their own suboptimality or focus on characterizing and investigating the suboptimalities of existing algorithms. There is no consistent benchmarking methodology when comparing sizing heuristics. Results in [21] show that sensitivity-based and continuous solution-guided approaches are not robust, as their suboptimality varies widely (from 4% to 52%) when applied to different ISCAS-85 benchmarks having nearly identical sizes. A more rigorous approach is needed to characterize and provide insight into the behavior of different algorithms over different classes of input circuits. Suboptimality studies of existing heuristics have already been performed for other VLSI problems such as logic synthesis [23], placement [24], [25] and optimal buffer insertion [26], [27]. However, to our knowledge we are the first to investigate the suboptimalities of gate sizing heuristics in a systematic way.

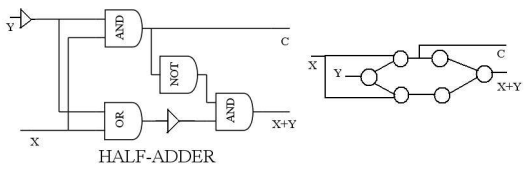


Fig. 2. The half-adder with buffers, with its corresponding graph shown on the right, is similar to a multi-output mesh topology, which is an extended version of the basic mesh topology, and is discussed in Section II-C.

In this paper, we present a method to generate combinational logic circuits (called *eyecharts*) which are combinations of the basic *chain, mesh* and *star* topologies shown in Figure 1. These eyecharts, together with their optimal solutions, can

be used to benchmark gate sizing heuristics. The idea is to test *just* gate sizing heuristics and not any structural/logic optimization heuristics. Our contribution is especially useful in benchmarking the heuristics which are often used during post-layout optimization phase where gate sizing, V_t-assignment, gate-length biasing are the main choices. We note that the basic eyechart topologies in Figure 1 represent the common elements of real circuits. For example, Figure 2 shows a half-adder circuit which has similarities to a multi-output mesh topology. Our benchmarks can be generated with various complexities and topologies (in terms of fanout, logic depth, and the number of primary inputs (PIs) and primary outputs (POs)) to study the behavior of existing algorithms under such variations. This helps us identify the weaknesses of existing algorithms and may help predict the behavior of a given heuristic or algorithm for a given arbitrary circuit. Our experiments show that the suboptimalities of popular sizing methods can be as large as 54%.[1]

The contributions of our paper are:

- a set of basic combinational logic topologies that we call eyecharts;
- a method to size the gates in the eyecharts optimally using DP;
- a method to form arbitrarily large combinational logic circuits by daisy-chaining the proposed basic topologies, while retaining the ability to optimally size these circuits; and
- experiments and results that show the suboptimality behavior of five commonly used gate sizing methods (including two commercial tools) under varied topological, delay/power modeling as well as optimization contexts.

The organization of our paper is as follows. Section II describes our method to optimally solve the basic and hybrid eyecharts. Section II-B describes the method to construct and solve larger *hybrid eyecharts* (formed by daisy-chaining the basic eyecharts). Section III describes the implementation details and the experimental setup to study suboptimalities of the compared heuristics. Section IV reports the results of several suboptimality case studies. In Section V, we extend our dynamic programming approach to solve eycharts for a more complex delay model where the delay of a gate depends on both input slew and load capacitance. In the same section we present the corresponding experimental setup and results.

II. Solving Eyecharts Optimally

In this section, we present a method to perform optimal sizing of the basic eyechart topologies shown in Figure 1 using DP. We assume that a gate's delay depends only its size and the total load capacitance.

A. Solving Basic Eyecharts

Solving a Chain. Optimally solving a chain topology entails allocation of delay budget to each *stage* such that the total power is minimized. Here, stage refers to the level of the logic gate, with PIs at the first level or stage. The delay budget assignment (without output load dependence) is essentially a multi-stage allocation problem which can be solved optimally using DP [28].

For an N-stage chain, the DP recursion is shown in Equations 1 and 2. We assume that a gate's delay depends only on its size and its total output capacitance. We assume that each gate has k discrete sizes. D_{max} denotes the maximum delay constraint, $C(s)$ is the input capacitance of an inverter of size s, p_{ij} and $d_{ij}(C(s))$ respectively indicate the leakage power and delay (for an output capacitance $C(s)$) of the gate at stage i with size j. Let x_i denote the cumulative delay budget for stage i; then the total power (for an output load $C(s)$) through stage i is denoted by $P_{i,C(s)}(x_i)$.

$$P_{1,C(s)}(x_1) = \min_j\{p_{1j}\} \text{ s.t. } d_{1j}(C(s)) \leq x_1,\ 1 \leq j \leq k,\ 1 \leq s \leq k \quad (1)$$
$$\min_j\{d_{1j}(C(s))\} \leq x_1 \leq D_{max}$$

$$P_{i,C(s)}(x_i) = \min_j\{p_{ij} + P_{i-1,C(j)}(x_i - d_{ij}(C(s)))\} \text{ s.t. } d_{ij} < x_i \quad (2)$$
$$1 \leq j \leq k,\ 1 \leq s \leq k$$
$$\min(x_{i-1}) + \min_j\{d_{ij}(C(s))\} \leq x_i \leq D_{max},\ \text{for } i \leq 2 \leq N-1$$
$$\text{and } x_N = D_{max}$$

For any stage i, the optimal size for a given cumulative delay budget is determined by an exhaustive local search among the available gate sizes. This is done for all of the possible output loads seen by stage i due to stage $i+1$. Stage $i+1$ is then solved optimally by considering all of its gate size choices and the stage i's optimal size (for an output load equal to stage $i+1$'s input capacitance) for the range of cumulative delay budgets shown in Equations 1 and 2. It is easy to see that the problem has an optimal substructure.[2] Dynamic programming therefore solves this sizing problem optimally. Note that the principle of optimality holds *only* if the delay of stage i depends only on the input capacitance of stage $i+1$. This "levelization" of the circuit graph is the key for preserving optimality.

We observe that for each stage, an optimal gate size entry exists for all possible cumulative delay budgets and for all possible output loads. The last stage's cumulative delay budget table has only one entry corresponding to the given PO load capacitance and the maximum delay constraint (since optimal sizing uses the full delay budget). A DP execution for a three-stage inverter chain, with the delay model shown in Table I, is shown in Table II (CP denotes cumulative power, OS denotes optimal gate size).

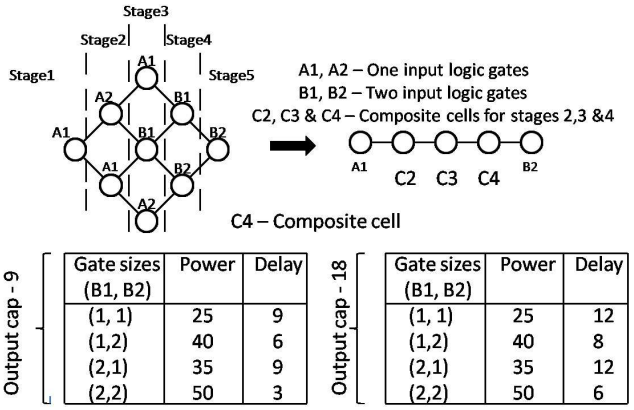
After constructing the table, the optimal size for each stage is found by traversing the cumulative delay budget table backwards from PO to PI and allocating a delay budget (and therefore a size) to each stage. The bolded values in Table II show this.

[1]Without loss of generality, we only present the details for leakage optimization in this work.

[2]Consider an n-stage chain not containing the optimal solution for $n-1$ stages with the given output load. In that case, the solution can be improved by substituting the optimal solution for stage $n-1$.

TABLE I
DELAY TABLE FOR THE INVERTER USED IN THE NUMERICAL EXAMPLE.

	Input cap	Leakage power	Delay	
			Load cap 3	Load cap 6
Size 1	3	5	3	4
Size 2	6	10	1	2



Output cap - 9

Gate sizes (B1, B2)	Power	Delay
(1, 1)	25	9
(1,2)	40	6
(2,1)	35	9
(2,2)	50	3

Output cap - 18

Gate sizes (B1, B2)	Power	Delay
(1, 1)	25	12
(1,2)	40	8
(2,1)	35	12
(2,2)	50	6

Fig. 3. Mesh to chain reduction. Delay and power values for B1 and B2 are assumed to be twice and thrice, respectively, that of the values shown in Table I.

Solving Mesh and Star Structures. Mesh and star topologies are optimally solved by first reducing them to chains. This is shown in Figure 3 and Figure 4. A stage with two or more gates is represented using a *composite cell*, to capture the power and delay characteristics of all the gates in that stage. A composite cell of a stage is a tabular representation that has power and delay values for all the possible gate size combinations of all the gates belonging to that stage, for all the possible output load combinations. For example, if A1, A2, B1 and B2 each have two gate sizes, then the composite cell of stage 3 will have $2^3\{A_1, B_1, A_2\} \times 2^2\{B_1, B_2\}$ entries. Since each stage with multiple gates is enumerated for all the possible output loads, *we preserve optimality in this reduction even if the gates belonging to a stage are non-homogeneous.* The gate size entry and the input capacitance entry of a composite cell is a vector of gate sizes and input capacitances respectively, of each gate in that stage. For a gate size of a composite cell, the power is the sum of the powers and the delay is the maximum of the delays, of the individual gates. An example of this is shown for stage 4 of the mesh topology in Figure 3. The composite cell *C4* represents the gate size vector, power and delay values of the gates of stage 4.

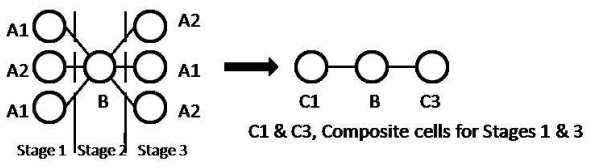


Fig. 4. Star to chain reduction.

B. Solving Hybrid Eyecharts

We generate large hybrid eyecharts by daisy-chaining the basic chain, mesh and star topologies. In this section, we explain how to optimally solve the hybrid eyecharts. We solve a hybrid eyechart by reducing it to a simple chain using a process similar to the one described in Section II. Figure 5 shows a sample hybrid eyechart and its reduced chain equivalent. While creating a hybrid eyechart, we mark each gate in the circuit with a tag that indicates whether it belongs to a chain, mesh or star structure. All the stages that belong to a mesh are reduced to single cells using composite cell models. Starting from each PI, we build cumulative delay budget tables for each stage until the star node in the center (gate C in Figure 5) is reached. For each such PI chain, the cumulative delay budget table of the last stage has cumulative power values for different cumulative delay budgets for the whole PI chain. Now, these PI chains in parallel can be represented using a single composite cell, as with Chains 1 and 2 in the hybrid eyechart of Figure 5. The entries for the power of this cell for each budget will be the sum of the individual powers of each PI chain.

Budget tables are built for each stage of the Chains 3 and 4 by following the same steps taken for the PI side chains, allowing them to be represented using a single composite cell. The whole circuit can now be treated as a chain and solved using the DP recursion described in Section II. To determine the optimal sizes for all the gates in the circuit, the delay budgets allocated to each cell of the reduced chain are applied to the corresponding PI/PO chains. The stages with only one gate are assigned gate sizes directly from the cumulative budget table, and the gate sizes for other stages are assigned from the entries found in their respective composite cell tables.

TABLE II
NUMERICAL EXAMPLE FOR A THREE-STAGE INVERTER CHAIN . THE FINAL OPTIMAL SIZING SOLUTION IS SHOWN IN **bold** FONT.

Output cap	Stage 1			Stage 2			Stage 3		
	Budget	CP	OS	Budget	CP	OS	Budget	CP	OS
3	**1**	**10**	**2**	3	20	2			
3	2	10	2	**4**	**15**	**1**			
3	3	5	1	5	15	2			
3	4	5	1	6	10	1			
3	5	5	1	7	10	1			
3	6	5	1	8	10	1			
3	7	5	1						
3	8	5	1						
6	2	10	2	4	20	2	**8**	**20**	**1**
6	3	10	2	5	15	1			
6	4	5	1	6	15	2			
6	5	5	1	7	10	1			
6	6	5	1	8	10	1			
6	7	5	1						
6	8	5	1						

C. More General Eyechart Topologies

Topologies that are more complex than the basic eyechart topologies can also be solved. We illustrate this with a multi-output mesh topology, as shown in Figure 6. This topology has two two-input cells added in the last stage. A unique property of the three basic eyechart topologies, unlike the multi-output mesh, is that a gate at any stage i has inputs coming only from stage $i-1$ and its fanout goes only to gates in stage $i+1$. We treat the group of stages which do not satisfy this unique

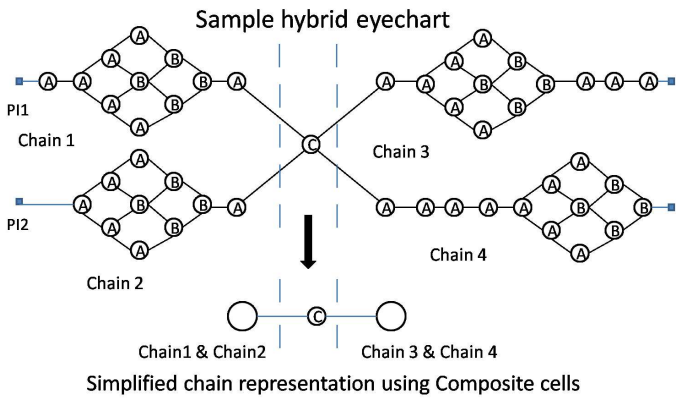


Fig. 5. Hybrid eyechart reduced to a chain.

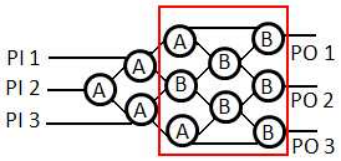


Fig. 6. Multi-output mesh topology.

property as a single stage and solve it using local enumeration. For example, in Figure 6, stages three, four and five are treated as a single stage (enclosed by a box), which is enumerated. The rest of the circuit is solved using the method described in Section II. Other similar topologies can be added (albeit at the cost of optimization runtime) in the same way, which can make our eyechart approach fairly flexible.

III. Implementation Details and Experimental Setup

A C++ program has been written which generates and solves these eyecharts, and which also writes out their corresponding netlists, parasitics and delay models in the industry-standard .v, .spef and .lib formats so that they can be easily used with standard sizing tools. The complexity of the DP recursion described in Section II is bounded by $O(P \cdot N \cdot B \cdot k^n)$, for a circuit with P PIs/POs, N stages, delay budget B, k sizes per gate and n gates per stage. Solving a hybrid benchmark circuit with 10,000 gates takes approximately four hours.[3]

Delay tables are formed with delay entries for all possible output load combinations, to avoid interpolation. This makes the number of capacitance indices for the delay tables dependent on the maximum fanout in the circuit. For example, if the library has three sizes per standard cell and the maximum fanout in the circuit is three, then the delay table would have nine capacitance indices. This ensures that any combination of output loads will fall in the range of available discrete capacitance entries in the delay table. To test the suboptimalities of post-routing leakage optimization tools, we also insert parasitic capacitances at different nodes. The values of the parasitic capacitances are set to be integral multiples of a minimum constructed gate capacitance so that the total load capacitance seen by a gate has a corresponding index in the delay table. To preserve optimality, we ignore some complexities (e.g., crosstalk, slew propagation, etc.), which are handled by practical optimization engines.

We experimented with the following five different gate sizing methods for suboptimality studies.

- *Comm1, Comm2*: Two well known commercial gate sizing and leakage optimization tools. Unfortunately, we do not know the internal details of these optimizers.
- *LP*: A linear programming-based slack allocation and sizing tool which is an implementation of [8]. First, it optimizes the circuit for maximum speed. Then, it uses power-delay sensitivities in a linear programming-based slack allocation to maximize the power savings. It uses a freely distributed linear programming solver *lp_solve* (http://lpsolve.sourceforge.net/5.1).
- *GS*: A greedy sensitivity-based sizing tool similar to [3], [17] but with a TILOS [4]-like sensitivity function ($\Delta power/\Delta delay$).
- *SBS*: A sizing tool that uses the slack-based greedy sensitivity metric ($\Delta power/\Delta slack$) proposed in [3], [17].

Note that all the above heuristics except LP use an incremental timing engine in the optimization loop. Due to delay modeling approximations, LP can sometimes violate timing constraints. We pick the delay constraint values (by trial and error) that LP can achieve and run the rest of the heuristics (as well as the optimal DP) with those constraints. This ensures that all our comparisons are fair in terms of suboptimality, but a practical LP-based optimizer will need additional hooks to fix timing violations.

The following four types of delay and power tradeoffs with size are investigated.

- LP-LD: Linear increase in power with size and a linear fit of delay to size/load.
- LP-NLD: Linear increase in power with size and a nonlinear fit of delay to size/load.
- EP-LD: Exponential increase in power with size and a linear fit of delay with size/load.
- EP-NLD: Exponential increase in power with size and a nonlinear fit of delay to size/load.

In the EP-LD and EP-NLD models, we use the term "cell-variant" to indicate that the cell swapping choices are V_t or gate-length variants. We assume that the input capacitance of a gate remains the same across V_t variants. For gate-length variants, the capacitance increases linearly with gate-length. All the delay values were fitted individually for each type of standard cell, using an industrial multi-V_t 65 nm CMOS technology library. Table III gives a summary of the characteristics of the four library models used in our experiments, along with the corresponding optimization contexts.

Table III shows the RMS error of each of these fits. Power values for all four library models are taken from the reference

[3]Note that this runtime is not a huge concern since the DP method is not intended for use in practical optimization, but only for benchmarking.

technology library and hence do not involve fitting (except for the studies which involve more number of sizes/variants than the listed default number). The minimum delay budget for any benchmark is found by maximally sizing all the gates in the design.[4]

TABLE III
SUMMARY OF LIBRARY MODEL CHARACTERISTICS.

Library model	RMS fitting error (delay)	Optimization context	Default # sizes/variants
LP-LD	8.43%	Gate sizing	8
LP-NLD	0.3%	Gate sizing	8
EP-LD	8.43%	V_t , gate-length bias	3, 3
EP-NLD	0.3%	V_t , gate-length bias	3, 3

IV. SUBOPTIMALITY CASE STUDIES

In this section we present the results of a few interesting case studies. We compare the suboptimalities of the five optimization heuristics outlined in Section III. These studies are by no means exhaustive and many other experiments are possible using eyecharts as a diagnostic tool. All of the suboptimality values are calculated as

$$\text{Suboptimality}\% = \frac{\text{method\_power} - \text{optimal\_power}}{\text{optimal\_power}} \times 100. \quad (3)$$

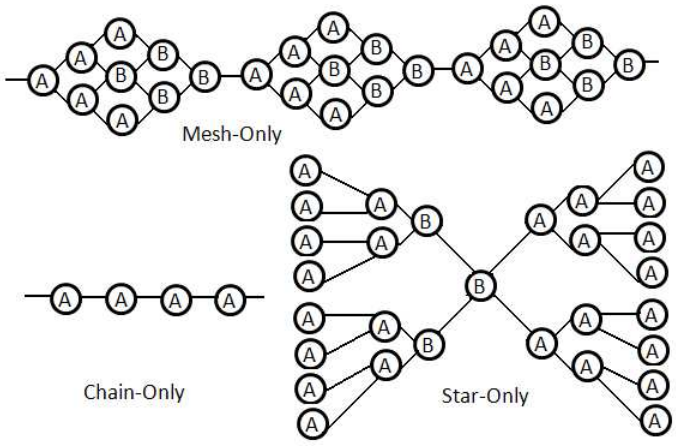


Fig. 7. Daisy-chained individual topologies.

A. Dependence on Circuit Topology

Figure 7 illustrates large chain-, mesh- and star-only circuits obtained by daisy-chaining the basic topologies. We fix the netlist size at approximately 10,000 gates and perform the optimizations with varying delay constraints. The average fanin/fanout depends on the length of the mesh (three-stage, five-stage etc., mesh), maximum fanin of the star-only cells, and total number of stages in the daisy chain. The average fanouts (and fanins, due to symmetry) of chain-, mesh- and star-only topologies used in this experiment and shown in Figure 7 are 1, 1.875 and 1.95 respectively. Figure 8 shows the suboptimalities for mesh and star topologies; these results indicate that a mesh structure is more difficult to solve than a star structure. With respect to circuit topology, *designers should look out for mesh-like topologies (i.e., with reconvergent fanouts) since none of the tested heuristics perform well on them.*

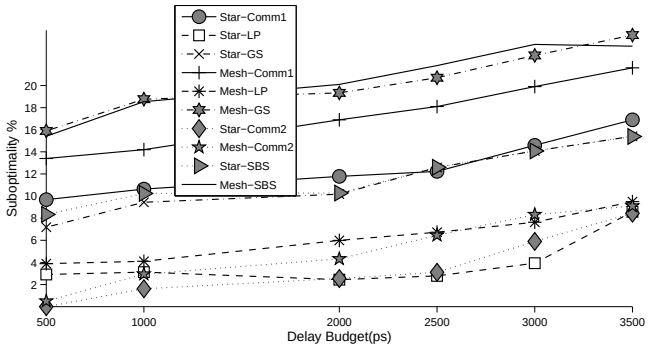


Fig. 8. Suboptimality comparison of the daisy-chained mesh-only and star-only topologies. The chain-only suboptimality results show that all the compared methods have close to zero suboptimalities and hence are not shown.

We also studied the impact of circuit size on suboptimality. We varied the circuit's logic depth and number of PIs/POs, while keeping the topology of the hybrid eyechart the same as in Figure 5. The corresponding results (on benchmarks with sizes ranging from 100 to 51,500 gates) show that the suboptimality is unaffected by the size of the design, and hence for the rest of the studies, we use a hybrid benchmark with 10,000 gates.

B. Effect of Delay Modeling

We compare the delay models using a single hybrid benchmark with 10,000 gates. This benchmark has a topology similar to the sample hybrid eyechart of Figure 5 with a multi-output mesh topology added at each of the POs. It has four PIs and five POs. 1,000 five-stage mesh topologies and inverter chains are also inserted randomly into the benchmark. INV, NAND and four-input AOI gates, each with eight discrete gate sizes, are used.

We experimented with the LP-LD delay model and the suboptimality results are shown in Figure 9. *Comm2* has the best suboptimality, while *LP* performs much better than the three remaining optimizers. This is expected since linear power-delay tradeoffs are better suited for the LP-based slack-allocation engine. To evaluate the performance of the chosen optimizers with realistic delay models, .lib models are generated with the LP-NLD delay model. Figure 10 shows the corresponding suboptimality trends. Note that the delay dependence on size is only weakly nonlinear in the 65 nm library used in this paper. The performance of the *LP* and *Comm2* methods suffers under the nonlinear delay model while that of *Comm1*, *SBS* and *GS* improves, especially with relaxed timing constraints. A possible reason is the ability of *Comm1*,

[4]Note that this may not be the fastest possible implementation for the case of gate sizing, where the dependence of the circuit delay on individual gate sizes is not necessarily monotone.

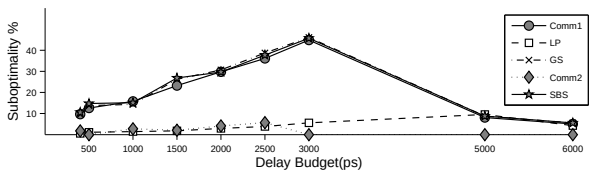


Fig. 9. Suboptimalities for LP-LD model.

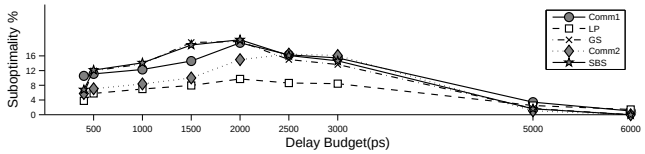


Fig. 10. Suboptimalities for LP-NLD model.

SBS and *GS* to exploit the nonlinearity. Without knowing the details of *Comm1* and *Comm2*, one conclusion we can draw is that even with the nonlinearity in the commercial delay tables, *linear programming is a good approach for linear power tradeoffs.*

The non-monotone trends in the suboptimalities can be explained using Figure 11. The *suboptimality of any of the compared methods is the least for both the smallest and the largest delay budgets, while the optimal power decreases with increasing delay budget.* This results in the observed non-monotone behavior of the suboptimalities.

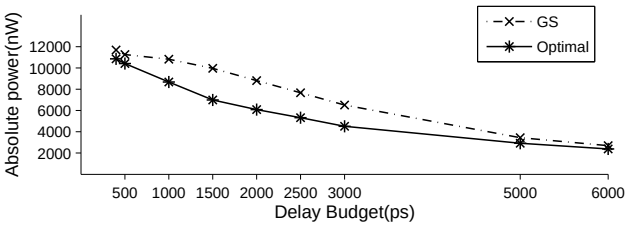


Fig. 11. Absolute power comparison for LP-LD model.

C. Effect of Library Granularity

We experimented with a varying numbers of gate sizes for the LP-NLD model. For these experiments, the delay and power ranges are kept the same as in Section IV-B while the granularity in gate sizes is increased. Figure 12 shows the corresponding trends in suboptimality. It can be seen that the performance of *Comm1* and *Comm2* are relatively unaffected while *LP's suboptimality improves noticeably, which is likely due to a smaller error in snapping the continuous solution to a discrete one.*

For the same experiment, Figure 13 shows the optimal power values for different delay budgets with two, four and six gate sizes, normalized to the optimal power values with eight gate sizes. Since the delay range is the same for these experiments, higher granularities create more gate sizing options for smaller delay increments. Hence, *a higher library granularity results in a lower optimal power, but the difference is not very pronounced.*

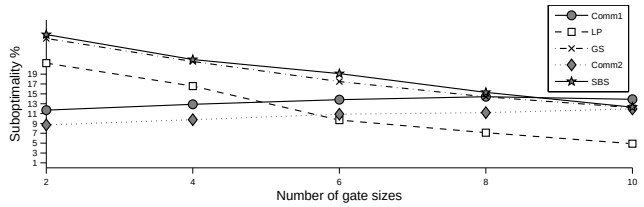


Fig. 12. Suboptimalities vs. number of gate sizes for LP-NLD model with 1500 ps delay budget.

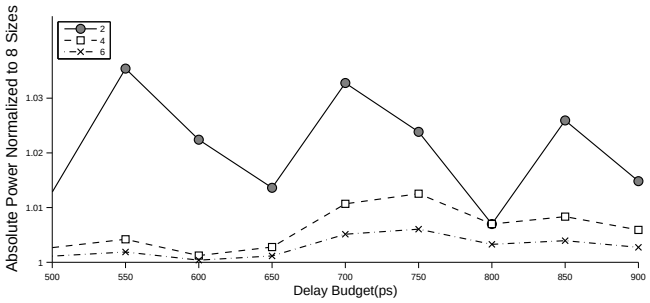


Fig. 13. Optimal power vs. delay budget for different gate size granularities. Power values are normalized to corresponding optimal results for eight gate sizes.

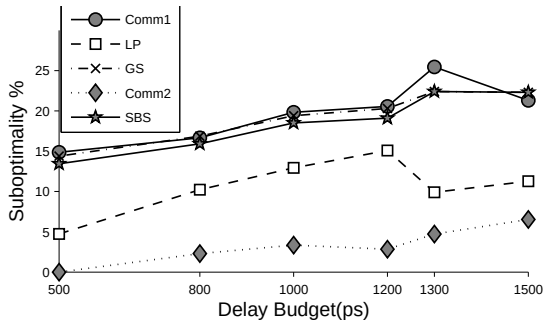


Fig. 14. Suboptimalities for EP-LD model under different delay constraints, using V_t variants only.

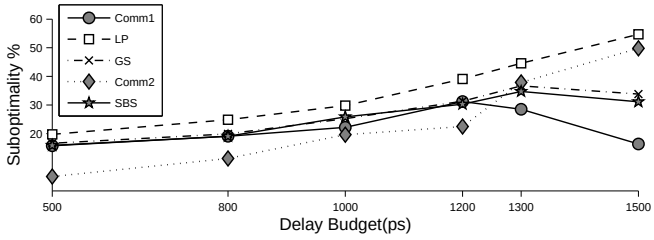


Fig. 15. Suboptimalities for EP-NLD model using V_t variants only.

D. The V_t-assignment Context

We experimented with the EP-LD and EP-NLD library models by varying the delay constraint for the case where each cell has three variants. The corresponding results are shown

in Figure 14 and Figure 15 respectively. The minimum delay budget for optimization is found by assigning low V_t variants to all of the cells. *LP* and *Comm2* perform relatively well for the smaller budgets but their suboptimality increases for the larger delay budgets. Moreover, the suboptimality difference between the LD and NLD delay modeling is more pronounced for the *LP* optimizer.

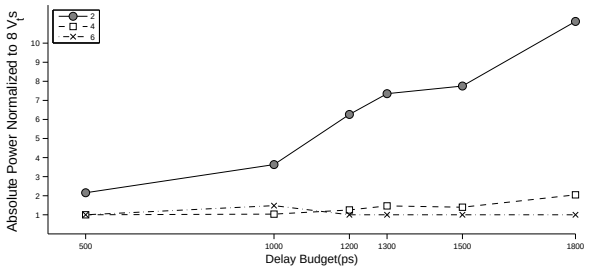


Fig. 16. Optimal powers for varying delay budgets and different V_t granularities. The powers are normalized to the corresponding results for eight V_t variants.

Figure 16 shows the optimal power values for different delay constraints for two, four and six V_t variants, normalized to the powers with eight V_t variants.[5] Since the delay range is the same for these experiments, higher granularities create more V_t options for smaller delay increments, resulting in lower optimal powers. These results suggest a *strong benefit in increasing the number of cell variants when the power tradeoff is exponential*, in contrast to the case in Figure 13, which has a linear power tradeoff.

E. The Gate-length Biasing Context

Figure 17 shows the trends for gate-length biasing. Gate-length variants have different input capacitance values as opposed to V_t variants. In these results, all of the tools perform worse than the gate sizing or V_t-assignment cases. This is especially true for *LP* whose suboptimality rises to over 40% and it does not perform better than the simple GS and SBS approaches. Iterative methods like *GS* and *SBS*, which use a real static timing engine, perform better than *LP* due to the near-quadratic dependence of the delay on the gate-length. This near-quadratic dependence is due to the reduced drive strength of the driver, and the increased input capacitance of the load, for an increased gate-length.

F. Observations

We summarize our main observations and conclusions below.

- All of the tools fare well on designs with low fanouts (i.e., designs that are topologically close to the chain eyechart).
- All of the tools fare poorly on designs with reconvergent fanouts (i.e., designs that are topologically similar to the mesh eyechart).

[5]We realize that more than three V_ts are rarely used in practice. Nevertheless, we show the results with more variants to highlight what is achievable.

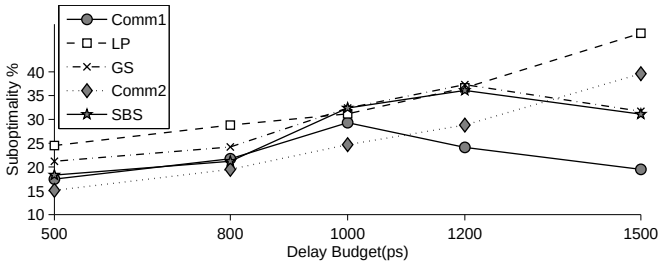


Fig. 17. Suboptimalities for EP-NLD model using gate-length variants only.

- For gate sizing and V_t-assignment, linear programming-based solvers can perform surprisingly well. Their solution quality suffers significantly in the gate-length biasing context where the delay is a strongly nonlinear function of the gate-length.
- The commercial tools do well in different optimization contexts (but unfortunately we cannot offer more insight here). These tools, as well as the sensitivity-based sizers *GS* and *SBS*, benefit from not needing to fit a closed-form delay/power model to real library data.
- The benefits of having a finer granularity in a library is much more pronounced for exponential power tradeoffs (such as gate-length biasing and V_t-assignment) than linear power tradeoffs (as in the case of gate sizing).
- Local sensitivity-based heuristics like *GS* and *SBS* can be highly suboptimal for large delay budgets due to their tendency to be trapped in local minima (as we have seen with star or mesh topologies).

V. Extensions to Complex Delay Models

In this section, we present our method to solve the basic and hybrid eyecharts for a more complex delay model. The delay of a gate is assumed to depend on its size, input slew and total load capacitance. The output slew of a gate is assumed to depend on its size and the total load capacitance.[6] **Our method does not guarantee optimality under this delay model but still produces significantly better solutions than any of the sizing heuristics we have evaluated.**

A. Solving Eyecharts

Solving a Chain. Solving a chain for this assumed delay model involves a method similar to the one described in Section II-A except that we build cumulative delay budget tables for all possible output loads and also for all possible input slews. The result is a 3D cumulative delay budget table for each stage with input slew, output load and cumulative delay budget as the three indices, with the first stage having constant slew index corresponding to the input slew at the PI; and the last stage having a constant output load index corresponding to the output load capacitance at PO.

While building the cumulative delay budget table for an input slew for stage i, the cumulative delay budget of stage

[6]In this work, we ignore slew propagation (as do most published sizing methods) to preserve optimality or near-optimality of dynamic programming based methods.

$i-1$ whose output slew is consistent with the assumed input slew is chosen. But there could be multiple such cumulative delay budgets since stage $i-1$ is also built for different input slews.[7] So, we choose the one among such cumulative delay budgets for which cumulative power is least and store it in the table along with the slew index it belongs to (for ease of lookup while reverse propagating the tables).[8] While reverse propagating, the cumulative delay budget $Dmax$ of stage N with the least power contains the optimal solution (optimal gate size of stage N and optimal power of whole circuit). The stored values of $N-1$ stage's cumulative delay budget and slew index of stage are then used to continue with the reverse propagation.

Due to the need for maintaining slew and load consistency while building the tables, DP cannot guarantee an optimal solution. For example, optimal solution of a three-stage inverter chain may not contain the optimal solution for the first two stages since the optimal solution for two stages only need not enforce a given slew at the output. Even in the presence of this suboptimality, our experimental results show that existing methods used for comparison are still considerably further worse.

Slew consistency constraint while building cumulative budget tables introduces slight suboptimality if we build cumulative delay budget tables from PI to PO. We can also solve a chain in a reverse fashion by building cumulative delay budget tables from PO to PI and allocate budgets by traversing tables from PI to PO. In this case capacitance assumption consistency leads to slight suboptimality. We take the best of two optimization runs:

- run1 - build cumulative delay budget tables from PI to PO and allocate delay budgets from PO to PI; and
- run2 - build cumulative delay budget tables from PO to PI and allocate budgets from PI to PO.

Table IV (CP - Cumulative power, OS - Optimal size) shows *run1*[9] of DP execution for a three stage inverter chain, for an input slew of 2 and output load of 10. It shows how cumulative delay budget tables are built for all input slew and output load combinations for a maximum delay constraint of 8. Table V summarizes the assumed delay model. Sizes 1 and 2 have output slews 4 and 2 respectively.[10] Note that the cumulative delay budget table for first stage is built only for assumed input slew (which is 2 in this example). Also, for the final stage the cumulative delay budget table is built only for the assumed primary output load (which is 10 in this example).

Solving Mesh, Star and Hybrid Topologies. Since the delay of each gate of a stage depends on its input slew and output load, we first build composite cells for all input slew and output load combinations. It results in a 3D table with input slew, output load combination and current stage's gate size combination as indices. For example, for a stage with two gates (each with 3 gate sizes), 2 input slews and 4 output load combinations, the composite cell will have $2^3 \times 2 \times 4$ entries. The procedure to build a composite cell for an input slew and output load combination is the same as described in Section II-A. We also store the output slews of the stage in the composite cell model for ease of lookup. Once composite cells for each stage are built, mesh and star topologies can be reduced to chains and solved using the same procedure as described in Section II-A.

We solve hybrid eyecharts (shown in Figure 5) using a procedure similar to the one described in Section II-B. Each stage is identified as belonging to either a chain, mesh or star structure by the corresponding stored tag and composite cells are built accordingly. Using the composite cells the whole hybrid eyechart can again be treated as a chain and solved using the previously described procedure for solving a chain.

TABLE V
DELAY TABLE FOR THE INVERTER USED IN THE NUMERICAL EXAMPLE IN TABLE IV.

	Size 1		Size 2	
	Load cap 5	Load cap 10	Load cap 5	Load cap 10
Input slew 2	3	4	1	2
Input slew 4	5	6	3	4

B. Experimental Setup and Results

In this section we present the experimental setup and results of some of the experiments described in Section previously in this section for a chain. The experimental setup is similar to the one described in Section III. A C++ program has been written which generates and solves the eyecharts and also generates the corresponding netlists, parasitics and delay models in their respective industry standard formats. All the five heuristics mentioned in Section III are used for suboptimality study.

Delay tables are formed such that delay/slew entries are available for all possible input slew and output load combinations to avoid interpolation. Since output slew of a gate depends on its load capacitance, the number of slew and capacitance indices for the delay/slew tables becomes a function of the maximum fanout of the circuit. For example, if the library has 3 sizes per standard cell and the maximum fanout in the circuit is 3, then the number of capacitance entries required in the delay table would be 9. This can lead to 27 different possible slews (9 slews for 9 possible output caps, for each of the 3 sizes). This would make sure that any combination of existing gate sizes would fall in the range of the available discrete capacitance and slew entries in the delay table. However, the number of slew indices quickly rises with the maximum fanout of the circuit and the number gate sizes available. To keep the number of slew indices low, slew table entries are repeated. This is a realistic assumption since similar output slews for a gate of small size driving lower load capacitance and a gate of a higher size of the same

[7]This does not happen for i = 2, since stage 1's cumulative delay budget table is built for only one input slew.

[8]So we need to store stage $i-1$'s slew index only for $3 \leq i \leq N$.

[9]The *run2* of this example is executed in the same way except in reverse direction

[10]Due to space constraints, we assume one output slew per gate size to manage Table IV's size. In our experiments output slew for a gate depends on load capacitance also.

TABLE IV
NUMERICAL EXAMPLE FOR A THREE-STAGE INVERTER CHAIN . THE FINAL OPTIMAL SIZING SOLUTION IS SHOWN IN **Bold** FONT.

Input slew	Output cap	Stage 1				Stage 2					Stage 3					
		Budget	CP	OS	o/p slew	Budget	CP	OS	o/p slew	Stage1's budget	Budget	CP	OS	o/p slew	Stage2's budget	Stage2's slew
2	5	1	50	2	4	3	100	2	2	2						
2	5	**2**	**50**	**2**	**2**	4	100	2	2	3						
2	5	3	25	1	4	5	75	1	4	2						
2	5	4	25	1	4	6	75	1	4	2						
2	5	5	25	1	4	7	75	1	4	2						
2	5	6	25	1	4	8	75	1	4	2						
2	5	7	25	1	4											
2	5	8	25	1	4											
2	10	2	50	2	2	4	100	2	2	2	**8**	**125**	**2**	**2**	**6**	**2**
2	10	3	50	2	2	5	75	1	4	1						
2	10	4	25	1	4	**6**	**75**	**1**	**4**	**2**						
2	10	5	25	1	4	7	75	1	4	1						
2	10	6	25	1	4	8	75	1	4	2						
2	10	7	25	1	4											
2	10	8	25	1	4											
4	5					5	100	2	2	2						
4	5					6	75	1	4	1						
4	5					7	75	2	2	4						
4	5					8	50	1	4	3						
4	10					6	100	2	2	2	8	150	2	2	4	2
4	10					7	75	1	4	1						
4	10					8	75	2	2	4						

gate driving higher load capacitance are commonly seen in industry-standard libraries.

For these experiments also, we assume that input capacitance of a gate increases linearly with length for gate-length variants and remains constant across V_t variants. All the delay values are fitted for each type of standard cell, using the same industrial multi-V_t 65 nm CMOS technology library mentioned in Section III . Table VI gives a summary of the characteristics of the four library models used in our experiments along with the corresponding optimization contexts. The minimum delay budget for any benchmark is found by maximally sizing all the gates in the design.[11]

TABLE VI
SUMMARY OF LIBRARY MODEL CHARACTERISTICS.

Library Model	RMS fitting error (delay)	Optimization context	Default # sizes/variants
LP-LD	12.43%	Gate sizing	8
LP-NLD	0.6%	Gate sizing	8
EP-LD	12.43%	V_t , gate-length	3, 3
EP-NLD	0.6%	V_t , gate-length	3, 3

Chain-, star-, mesh-only experiments again showed that mesh is the most difficult topology. All the compared heuristics were close to the optimal solution for the chain-only case. Experiments with varying circuit sizes are also performed. The results are similar to the experimental results presented in Section IV and show that suboptimality is relatively constant with varying circuit size. So we chose a single hybrid benchmark with 10,000 gates and obtained the following experimental results.

[11]Note that this may not be the fastest possible implementation for the case of gate sizing, where the dependence of the circuit delay on individual gate sizes is not necessarily monotone.

LP-LD. We experimented with the LP-LD delay model and the suboptimality results are shown in the Figure 18. *Comm2* has the best suboptimality while *LP* performs much better than the other two optimizers. This is expected since linear power-delay tradeoffs are better suited for the linear programming based slack-allocation engine.

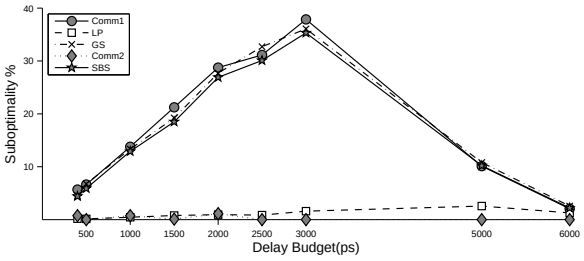


Fig. 18. LP-LD model suboptimality.

LP-NLD. To evaluate the performance of the chosen optimizers with realistic delay models where delay nonlinear tradeoff with size, .libs are generated with LP-NLD delay model. Figure 19 shows the corresponding suboptimality trends. Note that the delay dependence on size etc. is only weakly nonlinear in the 65 nm library we are working with. Performance of *LP* and *Comm2* methods suffers a bit under nonlinear delay model while the Comm1 and the simple GS heuristics improve significantly. Without knowing the details of *Comm1* and *Comm2*, one conclusion we can draw is that even with nonlinearity in commercial delay tables, *linear programming is a good approach at for linear power tradeoffs.* Similar to the results for LP-LD and LP-NLD experiments shown in Section IV, the suboptimalities show

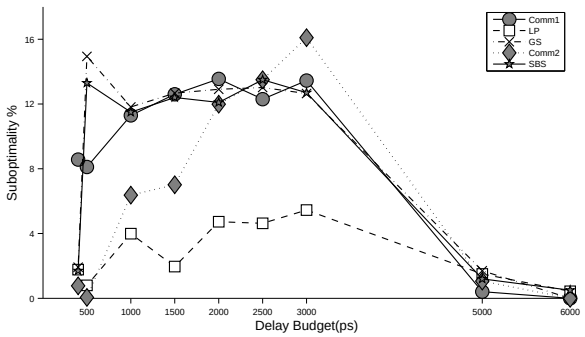


Fig. 19. LP-NLD model suboptimality.

a non-monotone behavior. Tools generate a solution close to the optimal for very tight or very relaxed timing constraints.

EP-NLD with V_t variants. We experimented with EP-NLD library model by varying the delay constraint for 3 cell variants and the corresponding suboptimalities are shown in Figure 20. *LP* and *Comm2* perform relatively well for lower budgets but tend be more suboptimal for higher delay budgets. Greedy methods *GS* and *SBS* also improve compared to LP-LD and LP-NLD case possibly due to the nature of greedy methods to exploit exponential tradeoffs.

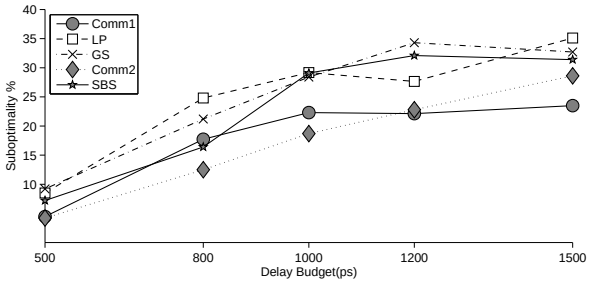


Fig. 20. Suboptimality for EP-NLD model using V_t variants only.

EP-NLD with gate-length variants. Figure 21 shows the trends for gate-length biasing context. Gate-length variants have different input capacitance values as opposed to V_t variants. All tools seem to do poorly here compared to gate sizing or V_t assignment optimizations. This is especially true for *LP* whose suboptimality rises to over 35%. Iterative methods like *GS* and possibly *Comm1* and *Comm2* all of which use a real static timing engine under the hood fare better than *LP* due to the somewhat quadratic dependence of delay on gate length (coming from reduced drive strength of the driver and increased input capacitance of the load with increasing gate-length).

VI. Conclusions and Future Work

We have described a method to generate arbitrarily large combinational circuits with known optimal solutions that can be used to benchmark and diagnose the problems with sizing heuristics. We have studied the optimization contexts of

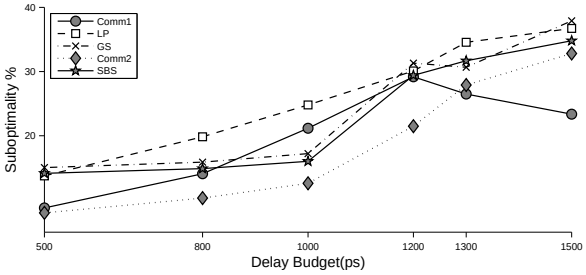


Fig. 21. Suboptimality for EP-NLD model using gate-length variants only.

gate sizing, V_t-assignment and gate-length biasing with two commercial and three academic sizing heuristics.

Our results show that these heuristics can be suboptimal by as much as 54%. The presence of reconvergent fanouts in a circuit topology results in a greater suboptimality compared to a similar topology without reconvergent fanouts. We also note that linear programming-based methods perform well under most scenarios, while local sensitivity (TILOS-like)-based heuristics perform better under exponential power-size tradeoffs. We have also proposed a method to solve eyecharts for a more realistic delay model where delay depends on load capacitance as well as input slew. Though optimality is not guaranteed in this case, the results show that the compared heuristics are significantly suboptimal. The benchmarks and the code can be downloaded from http://nanocad.ee.ucla.edu/Main/DownloadForm. Ongoing studies focus on generating large eyecharts according to a given fanout distribution, to create benchmarks that have characteristics close to real designs. Another interesting aspect of future work is a method to detect the presence of the basic eyechart topologies in real designs and measure the similarity of the designs to eyecharts with pre-characterized suboptimalities. This will enable the estimation of suboptimality of a given heuristic on a given design, possibly allowing optimization tools, which implement a collection of sizing heuristics, to choose one at runtime.

References

[1] P. Gupta, A. B. Kahng, A. Kasibhatla, and P. Sharma, "Eyecharts: Constructive Benchmarking of Gate Sizing Heuristics," *Proc. DAC*, 2010.

[2] W. N. Li, "Strongly NP-Hard Discrete Gate Sizing Problems," *Proc. ICCD*, pp. 468–471, 1993.

[3] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester, "Gate-length Biasing for Runtime-leakage Control," *IEEE Trans. on CAD*, pp. 1475–1485, 2006.

[4] J. Fishburn and A. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing," *Proc. ICCAD*, pp. 269–273, 1985.

[5] P. K. Chan, "Algorithms for Library-specific Sizing of Combinational Logic," *Proc. DAC*, pp. 353–356, 1990.

[6] M. R. C. M. Berkelaar and J. A. G. Jess, "Gate Sizing in MOS Digital Circuits with Linear Programming," *Proc. EURO-DAC*, pp. 217–221, 1990.

[7] K. Jeong, A. B. Kahng, and H. Yao, "Revisiting the Linear Programming Framework for Leakage Power vs. Performance Optimization," *Proc. ISQED*, pp. 127–134, 2009.

[8] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Tompson, and K. Keutzer, "Minimization of Dynamic and Static Power Through Joint

Assignment of Threshold Voltages and Sizing Optimization," *Proc. ISLPED*, pp. 158–163, 2003.
[9] A. Srivastava, "Simultaneous Vt Selection and Assignment for Leakage Optimization," *Proc. ISLPED*, pp. 146–151, 2003.
[10] H. Tennakoon and C. Sechen, "Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-based Pre-processing Step," *Proc. ICCAD*, pp. 395–402, 2002.
[11] C.-P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," *Proc. ICCAD*, pp. 617–624, 1998.
[12] S. S. Sapatnekar, V. B. Rao, and P. M. Vaidya, "A Convex Optimization Approach to Transistor Sizing for CMOS Circuits," *Proc. ICCAD*, pp. 482–485, 1991.
[13] K. Kasamsetty, M. Ketkar, and S. S. Sapatnekar, "A New Class of Convex Functions for Delay Modeling and its Application to the Transistor Sizing Problem," *IEEE Trans. on CAD*, pp. 779–788, 2000.
[14] H. Tennakoon and C. Sechen, "Efficient and Accurate Gate Sizing with Piecewise Convex Delay Models," *Proc. DAC*, pp. 807–812, 2005.
[15] O. Coudert, "Gate Sizing for Constrained Delay/Power/Area Optimization," *IEEE Trans. on VLSI Systems*, pp. 465–472, 1997.
[16] O. Coudert, R. Haddad, S. Manne, and S. Manne, "New Algorithms for Gate Sizing: A Comparative Study," *Proc. DAC*, pp. 734–739, 1996.
[17] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw, "Duet: an Accurate Leakage Estimation and Optimization Tool for Dual-Vt Circuits," *IEEE Trans. on VLSI Systems*, pp. 79–90, 2002.
[18] S. Hu, M. Ketkar, and J. Hu, "Gate Sizing For Cell Library-Based Designs," *Proc. DAC*, pp. 847–852, 2007.
[19] Y. Liu and J. Hu, "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment," *Proc. ISPD*, pp. 27–34, 2009.
[20] S. S. Shah, A. Srivastava, V. Zolotov, D. Sharma, D. Sylvester, and D. Blaauw, "Discrete Vt Assignment and Gate Sizing Using a Self-snapping Continuous Formulation," *Proc. ICCAD*, pp. 705–711, 2005.
[21] T.-H. Wu and A. Davoodi, "PaRS: Fast and Near-optimal Grid-based Cell Sizing for Library-based Design," *Proc. ICCAD*, pp. 107–111, 2008.
[22] H. Ren and S. Dutt, "A Network-Flow Based Cell Sizing Algorithm," *Proc. IWLS*, pp. 7–14, 2008.
[23] I. L. Markov and J. A. Roy, "On Sub-optimality and Scalability of Logic Synthesis Tools," *Proc. IWLS*, 2003.
[24] J. Cong, M. Romesis, and M. Xie, "Optimality and Stability Study of Timing-Driven Placement Algorithms," *Proc. ICCAD*, pp. 472–478, 2003.
[25] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng, "Quantified Suboptimality of VLSI Layout Heuristics," *Proc. DAC*, pp. 216–221, 1995.
[26] J. Lillis, C.-K. Cheng, and T.-T. Y. Lin, "Simultaneous Routing and Buffer Insertion for High Performance Interconnect," *Proc. GLSVLSI*, pp. 148–153, 1996.
[27] L. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," *Proc. ISCAS*, pp. 865–868 vol.2, 1990.
[28] R. Bellman, *Dynamic Programming*. Dover Publications, N.Y, 1957.