Construction of Realistic Gate Sizing Benchmarks With Known Optimal Solutions

Andrew B. Kahng
UC San Diego
La Jolla, CA 92093
abk@ucsd.edu

Seokhyeong Kang
UC San Diego
La Jolla, CA 92093
shkang@vlsicad.ucsd.edu

ABSTRACT

Gate sizing in VLSI design is a widely-used method for power or area recovery subject to timing constraints. Several previous works have proposed gate sizing heuristics for power and area optimization. However, finding the optimal gate sizing solution is NP-hard [1], and the suboptimality of sizing solutions has not been sufficiently quantified for each heuristic. Thus, the need for further research has been unclear.

In this work, we describe a new benchmark generation approach for leakage power-driven gate sizing (the subject of the forthcoming ISPD-2012 contest) which constructs realistic circuit netlists with known optimal solutions. The generated netlists resemble real designs in terms of gate count, maximum path depth, interconnect complexity (Rent parameter), and net degree distributions. Using these benchmark circuits with known optimal gate size, we have studied the suboptimality of several leakage-driven gate sizing heuristics, including two commercial tools, with respect to key circuit topology parameters. Our study shows that common sizing methods are suboptimal for realistic benchmark circuits by up to 52.2% and 43.7% for V_t-assignment and gate sizing formulations, respectively. The results also suggest that (1) commercial tools may still suffer from significant suboptimality, and/or (2) existing methods have "similar" degrees of suboptimality.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*Design Aids*

General Terms

Algorithms, Design.

Keywords

Benchmarks, Gate Sizing, Dynamic Programming, Leakage Power, Static Power, Optimization.

1. INTRODUCTION

The *sizing problem* in VLSI design seeks to assign design parameters (width and/or threshold voltage) to each gate, so as to optimize timing, area and/or power of the design subject to constraints. Gate sizing is widely applied during optimization and design closure phases of the implementation flow, since it helps to meet design constraints with minimal overall disruption. The problem has been extensively studied, and a number of heuristics have been proposed. Greedy heuristics for gate sizing to optimize power/area/delay subject to delay/area constraints are found in [2] and [3]. Sensitivity-based gate downsizing and V_t-assignment techniques are given in [4] and [5]. The sizing problem has been formulated as a linear program (LP) in [6] and [7]. Lagrangian relaxation (LR) based optimization is proposed in [8].

However, finding an optimal gate sizing solution is NP-hard [1], and the suboptimality of sizing solutions has not been quantified and analyzed sufficiently for available heuristics. Real circuits have unknown optimal solution quality, and thus do not shed much light on heuristic suboptimality. On the other hand, artificial circuits with known optimal solution quality – along with any implications they might have for suboptimality of heuristics – are viewed as unrealistic. Thus, the need for further research and development on gate sizing methods has been unclear. In this work, we focus on sizing for leakage reduction, and propose a new method for generating *realistic sizing benchmark circuits with known optimal sizing solutions*, which enables systematic and quantitative comparisons of available gate-sizing heuristics.

For evaluation of CAD heuristics, several methods of generating synthetic benchmarks that match real designs have been proposed. Darnauer and Dai [9] generate random benchmark circuits based on Rent's rule. Their code generates random circuits with a specified number of inputs, outputs, blocks, terminals per cell, and Rent parameter. Hutton et al. [10] define properties such as size, delay, physical shape, edge-length distribution and fanout distribution, and generate combinational circuits to match a given parameterization. Stroobandt et al. [11] provide parameterized (by Rent exponent and net degree distribution) benchmarks with user-selected library cells. With these synthetic benchmarks, various CAD heuristics can be compared to each other, but the suboptimality of the heuristics cannot be measured.

Suboptimality of existing heuristics has been studied for VLSI problems such as synthesis, placement, partitioning, and buffer insertion. Hagen et al. [12] show how to quantify the suboptimality of heuristic algorithms for NP-hard placement and partitioning problems arising in VLSI layout. They construct scaled instances from the original problem and execute the heuristic. If the heuristic solution cost increases at a faster rate than the scaling of the heuristic instance itself, this establishes a lower bound on the heuristic's suboptimality. PEKO (placement examples with known optimal solutions) [13] and its extension PEKU (placement examples with known upper bounds) [14] enable estimation of suboptimality of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISPD'12, March 25–28, 2012, Napa, California, USA.
Copyright 2012 ACM 978-1-4503-1167-0/12/03 ...$10.00.

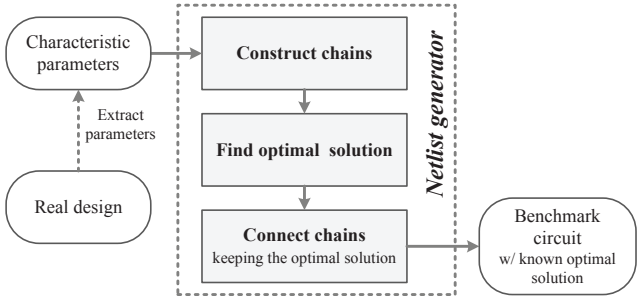


Figure 1: Generation of benchmark circuits with known optimal solutions.

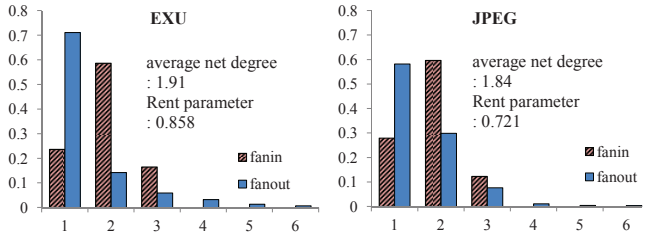


Figure 2: Circuit characteristics (fanin distribution, fanout distribution, average net degree and Rent parameter) for two real designs (EXU: OpenSPARC T1 execution unit; JPEG: JPEG encoder).

several timing-driven placement algorithms; the core approach involves perturbing an original design to obtain a new design with similar topological properties and a known optimal solution.

Our present work builds on the recent work of Gupta et al. [15], which to our knowledge is the only work in the literature to address suboptimality of (leakage-driven) gate sizing heuristics. The authors of [15] (1) propose *eyechart* benchmark circuits which can be optimally sized using dynamic programming methods, and (2) use eyecharts to evaluate the suboptimalities of several gate sizing algorithms. However, [15] does not address the difference or similarity between real designs and eyechart circuits. In [15], the *eyechart* circuits are built from three basic topologies – chain, mesh and star – and the resulting topologies differ substantially from those of real designs in terms of Rent parameter, path length and other parameters.[1] Thus, the eyecharts may be helpful in measuring suboptimality of heuristics, but do not have clear implications for heuristic performance on real designs. Furthermore, [15] does not provide any automated flow for eyechart circuit generation.

In this paper, we provide more realistic benchmarks with known optimal solutions for gate sizing problems. Figure 1 shows the flow of our benchmark circuit generation. (1) To create a circuit with known optimal gate sizing solution, we construct multiple chains (for which optimal sizing solutions can be found by dynamic programming), then connect the chains with inter-chain nets *without affecting the property of having a known leakage-optimal sizing solution*. (2) During the circuit construction, circuit topology is constrained according to user-specified parameters (path depth, and fanin / fanout distributions) so that the constructed benchmarks show similar characteristics to real designs. (3) The inter-chain connections can be added in many possible ways, which gives the potential for greater topological diversity than the previous construction of [15].

Our main contributions are summarized as follows.

- We propose benchmark circuits with known optimal solutions for gate (width and/or V_t) sizing, specifically, for leakage minimization subject to a (setup) delay constraint.
- The proposed benchmarks resemble real designs in terms of size, path depth (number of logic stages), interconnect complexity (Rent parameter), and net degree distribution. These parameters are extracted from real designs. The property of known optimal solution quality is maintained.
- We assess the suboptimality of standard gate sizing approaches, including two commercial tools, with respect to the above circuit parameters. Our results suggest that (1) commercial tools may suffer from significant suboptimality, and/or (2) existing methods have "similar" degrees of suboptimality for real designs.

[1]Eyecharts used in [15] have large depth (650 stages) and small Rent parameter (0.17). Table 4 below shows that real designs have path depths of $20 \sim 70$, and Rent parameter values of $0.72 \sim 0.86$.

The rest of this paper is organized as follows. Section 2 discusses how we address the two main considerations for gate sizing benchmarks – realism in circuit topology and tractability to optimal solution. Section 3 presents details of our benchmark generation procedure. Section 4 provides experimental results and analysis, including suboptimality studies of several heuristics and comparisons between real and artificial circuits. Section 5 summarizes and concludes the paper.

2. BENCHMARK CONSIDERATIONS

For benchmark circuit generation, *realism* and *tractability to analysis* are opposing goals since (1) determining the optimum solution is usually intractable in real designs, and (2) constructions for which optimum solution costs are known are often considered "artificial" [12]. We begin by considering this tension between realism and tractability in benchmark circuits.

First, to construct a *realistic benchmark*, we must use characteristic design parameters in the benchmark generation. Many works in the literature classify or parameterize circuits according to an empirical power-law scaling phenomenon that governs statistics of interconnects among and within subcircuits (cf. the well-known Rent parameter or Rent exponent [16]). Distributions of net degrees, or of the numbers of fanins and fanouts per cell instance, are additional important circuit characteristics. Figure 2 shows circuit characteristics of two real design blocks; each shows different characteristic parameters. In our work, to construct realistic benchmarks we use four design characteristic parameters: (1) number of primary (PIs and POs), (2) (maximum) path depth, (3) fanin distribution, and (4) fanout distribution. These four parameters can be configured in advance, and our benchmark generator makes net connections according to the given parameters subject to a given (setup) timing constraint.

Second, for generated benchmarks to permit *known optimal gate sizing solutions*, some simplifications are required. The eyechart work of [15] achieved tractable optimal solutions by simplifying the cell timing library to eliminate slew dependency. With such a simplified library, it is possible to find an optimal sizing solution for simple (chain) topologies using dynamic programming (DP). Star and mesh topologies can be reduced to equivalent chain topologies so that they, too, can be optimally sized using DP. In our present work, we use the same library simplification approach as [15]. However, we would also like to consider all possible topologies in order to satisfy our goal of realistic benchmark topologies; un-

fortunately, this makes optimal sizing intractable to DP even with the simplified timing library.

Our key insight is that instead of separating the netlist generation and optimization stages as in the eyechart approach, we can find optimal cell sizes *during* the benchmark netlist generation. We then augment the benchmark circuit without disturbing the existing, known optimal solution. More precisely: (1) we construct gate-chains to realize a specified number of primary input/output ports and a specified path depth; (2) we add fanins and fanouts to cells on the chains to match given fanin and fanout distributions; (3) we find optimal sizing solutions for cells in each chain using DP; and (4) finally, we connect the chains using *connection cells* while preserving the optimal gate sizing solution of each chain.[2]

3. BENCHMARK GENERATION DETAILS

Table 1 shows input parameters to our benchmark generation process. To simplify the procedure, we assume that the numbers of primary inputs and primary outputs are both equal to N. I and O respectively indicate the maximum numbers of fanins and fanouts to any given cell instance. Given the five input parameters, our flow generates N chains, each of which consists of K cells. We connect the chains using *connection cells* according to the prescribed fanin and fanout distributions. The result is a netlist with $K \cdot N + C$ cells, where C is the number of connection cells.

Table 1: Input parameters for benchmark generation.

parameter	description
T	timing path delay upper bound
N	number of primary inputs/outputs
K	(maximum) data path depth
$fid(i)$	fanin distribution (#cells with i = 1, ..., I fanins)
$fod(j)$	fanout distribution (#cells with j = 1, ..., O fanouts)

To generate the circuit properly, the input parameters must satisfy three constraints.

1. The timing budget T should be larger than minimum delay of a chain of K cells.
2. The total numbers of fanins and fanouts in the circuit should satisfy the equality of Equation (1).
3. The prescribed proportion of single-fanout cells, $fod(1)$, should be larger than the proportion of connection cells since connection cells have only one fanout.

We note that in real circuit designs (such as shown in Figure 2), fanout distribution tends to follow a power law, with $fod(1)$ typically greater than 0.6. Thus the third constraint above can be easily satisfied in realistic benchmarks.

$$\sum_{i=1}^{I} i \cdot fid(i) = \sum_{o=1}^{O} o \cdot fod(o) \quad (1)$$

[2]As discussed in [15], a leakage-optimal gate sizing solution can be known when the nonlinear delay model (NLDM) timing library (e.g., Synopsys .lib format) for the standard cells in eyecharts is modified to eliminate slew-dependence. Also, interconnect delays are omitted for simplicity. This departure from real performance libraries incurs the risk of misleading conclusions; thus, below we show comparisons made using real performance libraries (for which optimal solutions cannot be known).

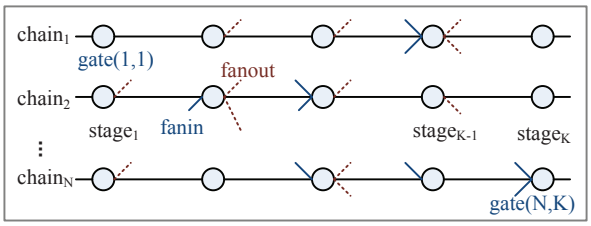
Algorithm 1 describes the procedure of benchmark generation. In the pseudocode, $gate(i,j)$ represents a gate at the j^{th} stage of the i^{th} chain. G_{co} is the set of connection cells. G_{fi} is the set of gate cells with open fanin ports. $DP(G_{chain}, T)$ is a dynamic programming procedure which finds an optimal cell sizing to minimize leakage power subject to the timing constraint T. We consider arrival times at the output side of any given gate, e.g., the arrival time at the output of gate g is denoted by a_g. Cell delay along the timing arc of cell g from the input that is connected to cell c is denoted by d_g^c. Finally, net delay along the net connecting c and g is denoted by $w_{c,g}$.

Algorithm 1 Netlist generation flow.

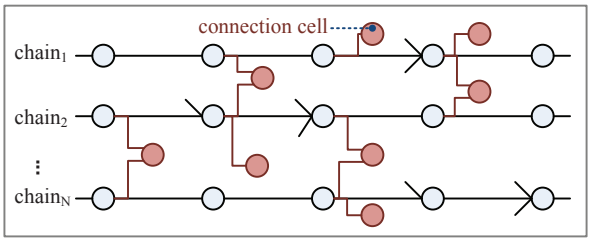
Procedure $NetlistGen(T, K, N)$
1. Initialize $gate(i,j)$, where $i = 1, ..., N$ and $j = 1, ..., K$;
2. $G_{co} \leftarrow \emptyset$, $G_{fi} \leftarrow \emptyset$;
3. **for** $j = 1 ; j \leq K ; j \leftarrow j + 1$ **do**
4. **for** $i = 1 ; i \leq N ; i \leftarrow i + 1$ **do**
5. Assign fanin number to $gate(i,j).fanin$;
6. Assign fanout number to $gate(i,j).fanout$;
7. **for** $k = 2 ; k \leq gate(i,j).fanout ; k \leftarrow k + 1$ **do**
8. Attach connection gate c to $gate(i,j)$;
9. $G_{co} \leftarrow G_{co} \cup \{c\}$;
10. **end for**
11. **if** $gate(i,j).fanin > 1$ **then**
12. $G_{fi} \leftarrow G_{fi} \cup \{gate(i,j)\}$;
13. **end if**
14. **end for**
15. **end for**
16. **for** $i = 1 ; i \leq N ; i \leftarrow i + 1$ **do**
17. $G_{chain} \leftarrow gate(i,j)$, where $j = 1, ..., K$;
18. $DP(G_{chain}, T)$; // find optimal gate size under T
19. **end for**
20. Update timing for all gates ($gate(*)$ and G_{co});
21. **while** $G_{co} \neq \emptyset$ **do**
22. Select gate c from G_{co} with maximum arrival time;
23. **for each** gate $g \in G_{fi}$ **do**
24. Select gate g with minimum arrival time;
25. **if** $a_c + w_{c,g} + d_g^c \leq a_g$ **then**
26. Connect c and g;
27. $G_{fi} \leftarrow G_{fi} - \{g\}$;
28. **break**
29. **end if**
30. **end for**
31. $G_{co} \leftarrow G_{co} - \{c\}$;
32. **end while**
33. Assign logic high or low to open input ports of $g \in G_{fi}$.;

First, we generate N chains, each with depth K (Lines 1 $\sim$ 15), as shown in Figure 3(a). For each of the $K \cdot N$ cells, we assign (i.e., instantiate) a gate according to the fanin distribution fid (Line 5). Cells in the first stage ($stage_1$) should be assigned one-input gates. Then, we assign the number of fanouts to the output of each cell (Line 6). Cells in the last stage ($stage_k$) have a single fanout. For remaining cells, the number of fanouts is assigned according to the fod. We have explored two alternative strategies for the fanin and fanout assignments: (1) *arranged assignment*, which assigns larger fanins to later stages and larger fanouts to earlier stages, and (2) *random assignment*, which assigns fanins and fanouts in arbitrary order. The arranged assignment improves connectability among the chains, while the random assignment improves diversity of the resulting topology.

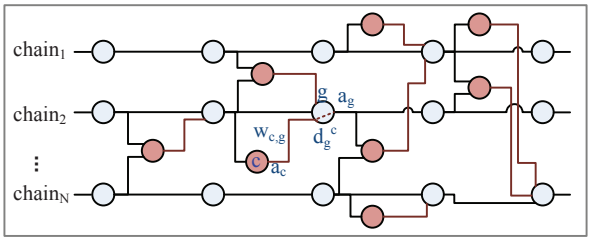
Second, we attach connection cells to open fanouts (Lines 7 $\sim$ 10), as illustrated by the red lines in Figure 3(a). The number of connection cells, C, is the same as the number of open fanin ports,



(a) Construct chains, and assign fanins and fanouts.



(b) Attach connection cells, and execute DP to determine optimal sizing.



(c) Connect chains subject to the (setup) timing constraint.

Figure 3: Netlist generation flow.

as expressed by Equation (2).

$$C = K \cdot N \sum_{i=1}^{I} (i-1) \cdot fid(i) \tag{2}$$

The fanin number of connection cells follows the fanin distribution (fid). For connection cells which have more than one fanin, open fanouts in the same stages are connected to the connection cell (Lines 11 $\sim$ 13), as illustrated in Figure 3(b).

After attaching all connection cells, we perform the dynamic programing (DP) with timing budget T for each chain (Line 18). The DP finds the optimal gate sizing which minimizes the leakage power for the chain. After the gate sizing, the sizes of attached connection cells will be set to minimum possible values since they do not have a timing constraint.

Finally, we connect all connection cells to any cells having open fanin ports (Lines 21 $\sim$ 32). Before connecting them, the arrival time for each cell is computed with static timing analysis (STA) with the timing budget T (Line 20). Connections between connection cells and open fanin cells are made only if the timing constraints are satisfied. In Figure 3(c), cell c and cell g can be connected when the arrival time of g via c ($a_c + w_{c,g} + d_g^c$) is less than the arrival time of g through the chain path. If timing slack of the connection cell is large, sizing heuristics can recognize them easily and the problem complexity will be the same as with a chain topology. To prevent this situation, we minimize timing slack of connection cells when making connections. Connection cells and open fanin ports are sorted according to their arrival time. Then, a connection is tried first between a connection cell with large arrival time and an open fanin port with small arrival time (Lines 22, 24). The connection cells do not change the optimal chain solution since they have minimum gate size. If we upsize them, there is no benefit to the timing slack of the main chain, and the optimal gate sizing of the chain does not change. Without timing constraints, our algorithm guarantees complete connection between open fanins and fanouts by virtue of Equation (1). With timing constraints, some ports can remain unconnected, which we address in Section 4.2 below. The open input ports are assigned with logic high (VDD) or low (VSS) according to the logic type. This assignment does not change the optimal solution.

After completing all the connections, we end up with a benchmark circuit of $K \cdot N + C$ cells with known optimal gate sizing for minimum leakage. (A small detail: when we use the generated circuit as a sizing benchmark, we initially assign maximum cell size (with highest leakage and fastest timing) to each instance, so as to avoid giving the leakage optimization tool any information about the optimal solution.)

4. EXPERIMENTAL SETUP AND RESULTS

4.1 Experimental Setup

Our netlist generator is implemented in C++ and produces a benchmark netlist in *Verilog HDL* ($.v$) with the corresponding delay models ($.lib$). Two types of delay and power models are used from the previous *eyechart* work [15][3] – (1) LP: linear increase in power with size for gate sizing context, and (2) EP: exponential increase in power with size for V_t or gate-length bias. The LP and EP power models have eight and three gate sizes (i.e., cell variants per master), respectively. To analyze the problem complexity of generated netlists, and suboptimality of standard sizing tools, we perform experiments on a 2.8 GHz Linux workstation with 24 GB RAM, using three different gate sizing methods – (1) two commercial gate sizing and leakage optimization tools (*BlazeMO v2008* [21] and *Cadence Encounter v9.1* [22]),[4] (2) a web-available *UCLA sizing tool* [25] (*Greedy*) which greedily swaps cells according to a $\Delta power/\Delta delay$ sensitivity function, and (3) a web-available UCSD sensitivity-based leakage optimizer [26] (*SensOpt*) with $\Delta power \times slack$ sensitivity function.[5] To generate realistic benchmark circuits, we use six open-source designs – *SASC* (asynchronous serial controller), *SPI* (serial peripheral interface), *AES* (data encryption), *JPEG* (image processing) and *MPEG* (video processing) from the *OpenCores* site [19], and *EXU* (execution unit) from *OpenSPARC T1* [20]. In our experiments, we measure the suboptimality of the various gate sizing heuristics, as is defined in Equation (3).

$$Suboptimality = \frac{power_{heur} - power_{opt}}{power_{opt}} \tag{3}$$

We use the same timing and power analysis tool (*Synopsys PrimeTime C2009.6* [24] to evaluate results.

[3]According to the authors of [15], EP corresponds to the multi-V_t context, and LP corresponds to the gate-length biasing context.

[4]These are referred to as *Comm1* and *Comm2* below. We do not give the mapping – i.e., which tool is *Comm1* and which is *Comm2* – in order to maintain anonymity as required by the tools' licenses.

[5]At the website [26], details of the UCSD *SensOpt* tool are given. The tool performs post-layout cell swapping using the *Tcl* socket interface to a golden STA tool, *Synopsys PrimeTime*.

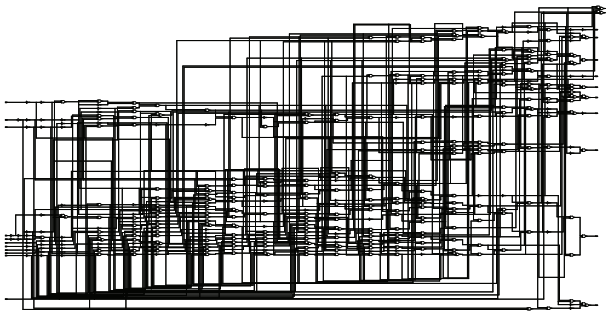


Figure 4: Schematic of generated netlist (N = 10, K = 20).

4.2 Generated Benchmarks

In this subsection, we present the results of generated benchmarks and their complexity in a power (leakage) optimization. Then, we compare the benchmarks and real designs in terms of characteristic parameters. Figure 4 shows the schematic of a generated netlist with 10 chains and path depth of 20. In the netlist, chains are connected to each other in arbitrary order, and various topologies can be found.

A connection between chains can be made when the newly generated path has positive (or zero) slack with respect to the timing constraint. As a result, some cells in the chain will have open ports and some connection cells will remain unconnected. If the number of unconnected cells is large, the generated netlist will deviate from the specified fanin and fanout distributions. As noted above, to improve the connectability we can assign the larger fanins to later stages, and larger fanouts to earlier stages. However, such an *arranged assignment* can reduce the difficulty of the sizing optimization for the benchmark: many connection cells will have loosely constrained timing (i.e., large slack), and this makes it easy to find the optimal solution. To consider both connectability and optimization difficulty, we mix the two alternative strategies – arranged and random assignments – in Algorithm 1, Lines 5 and 6. Table 2 shows the failure rate of connections among chains and problem complexity (suboptimality) according to the different mixtures of the arranged and random assignments. In the experiment, N and K values are fixed (40), and the EP power model is used. Suboptimality and runtime are obtained for the commercial tool (*Comm1*). The results show that 25% of arranged assignment in practice results in over 99% of connectivity, while also affording a sufficient problem complexity (11.2% suboptimality). The 100% random assignment shows smaller suboptimality (7.7%) for gate sizing because it results in many unconnected gates (17%). In all experiments reported below, we use the 75% random / 25% arranged assignment.

Since our benchmark generator makes chains first, then connects the chains to each other, we have assessed the problem complexity of benchmarks before and after the chain connection. Table 3 shows the suboptimality of leakage reduction for the commercial tool and the greedy method. The results show that the complexity (difficulty) of gate sizing increases with the number of chain connections. The chain-only structures are easy to solve, and heuristics show small suboptimalities ($\sim$3%). However, with added chain connections, the observed suboptimality (and inferred instance difficulty) increase significantly.

Table 4 shows the characteristic parameters of (a) real designs and (b) generated benchmarks. In the table, the Rent parameter has been evaluated using [18]. The real circuits do not follow the second constraint of our netlist generator (Equation (1)) since the numbers of primary inputs and primary outputs differ. For this reason, we select fanin and fanout distribution numbers that are only similar (not identical) to those of the real design when we perform the benchmark generation. From the results, generated circuits show similar design size, path depth, Rent parameter and average fanin (fanout); this offers hope that our benchmark generation approach can provide realistic benchmark circuits for gate sizing.

Table 2: Connectability and complexity (suboptimality) of generated netlists according to different proportions of arranged and random assignments.

arranged	random	unconnected	subopt.	runtime
100%	0%	0.00%	2.6%	108 sec.
75%	25%	0.00%	6.8%	97 sec.
50%	50%	0.25%	10.3%	120 sec.
25%	75%	0.75%	11.2%	225 sec.
0%	100%	17.0%	7.7%	311 sec.

Table 3: Instance complexities (difficulties) of chain-only and connected-chain topologies.

# of chain	# of stage	chain-only		connected	
		Comm1	*Greedy*	*Comm1*	*Greedy*
(a) EP library					
40	20	2.4%	0.3%	10.4%	8.7%
40	40	2.1%	1.3%	10.3%	11.1%
80	20	2.0%	0.5%	10.3%	10.9%
80	40	2.1%	1.3%	9.9%	10.9%
(b) LP library					
40	20	1.7 %	3.1%	7.7%	17.9%
40	40	2.4 %	3.5%	12.0%	18.5%
80	20	1.9 %	3.3%	12.3%	19.1%
80	40	2.5 %	3.5%	15.9%	19.6%

4.3 Suboptimality of Heuristics

In this section, we show suboptimality of standard heuristic solutions with our benchmarks. Figure 5 (respectively, Figure 6) shows suboptimality and runtime of heuristics (including *Comm1*) when the number of chains (respectively, number of stages) increases in the benchmark circuits.[6] From Figure 5, we see that the suboptimality increases slightly according to the design size, but runtime increases exponentially with the number of chains since total number of paths increases significantly with respect to the chain number. When the number of stages increases (Figure 6), the suboptimality increases especially for the *Comm1* and *SensOpt* solvers. We recall that the LP library model has a larger number (eight) of sizing candidates than the EP library model (three). We believe that the likely cause of the LP model showing a larger suboptimality and runtime for the greedy and sensitivity-based optimizations.

Figure 7 shows suboptimality and runtime results when the testcases have different topological complexities. To estimate the effect of netlist complexity, we change fanin and fanout distributions in the benchmark generation, such that each benchmark has a different average net degree. From the results, we see that subop-

[6]The same fanin and fanout distributions have been used for the experiments in Figure 5 and Figure 6 – fid: 0.3, 0.6, 0.1, fod: 0.6, 0.1, 0.2, 0.1 .

Table 4: Characteristic parameters of real designs and generated benchmarks.

testcase	path depth	#instances	Rent parameter	average fanin	fanin distribution			fanout distribution					
					1	2	3	1	2	3	4	5	6
(a) characteristic parameters of real designs													
SASC	20	624	0.858	2.06	0.169	0.606	0.225	0.663	0.177	0.045	0.029	0.015	0.071
SPI	33	1092	0.880	1.813	0.345	0.497	0.158	0.735	0.077	0.090	0.038	0.020	0.039
EXU	31	25560	0.858	1.91	0.237	0.587	0.165	0.711	0.142	0.059	0.032	0.014	0.007
AES	23	23622	0.810	1.89	0.237	0.637	0.126	0.694	0.120	0.062	0.039	0.026	0.015
JPEG	72	141165	0.721	1.84	0.280	0.597	0.123	0.582	0.299	0.077	0.011	0.005	0.004
MPEG	33	578034	0.848	1.59	0.334	0.567	0.04	0.681	0.244	0.021	0.007	0.003	0.002
(b) characteristic parameters of generated benchmarks													
ng_SASC	20	631	0.865	2.06	0.17	0.60	0.23	0.66	0.18	0.05	0.05	0.02	0.04
ng_SPI	33	1079	0.877	1.80	0.35	0.50	0.15	0.73	0.10	0.08	0.03	0.02	0.04
ng_EXU	31	24733	0.814	1.90	0.25	0.60	0.15	0.7	0.05	0.05	0.1	0.05	0.05
ng_AES	23	23780	0.820	1.88	0.24	0.64	0.12	0.7	0.05	0.05	0.12	0.03	0.05
ng_JPEG	72	132479	0.831	1.84	0.28	0.6	0.12	0.58	0.25	0.05	0.04	0.03	0.05
ng_MPEG	33	527995	0.848	1.60	0.42	0.56	0.02	0.68	0.2	0.04	0.03	0.02	0.03

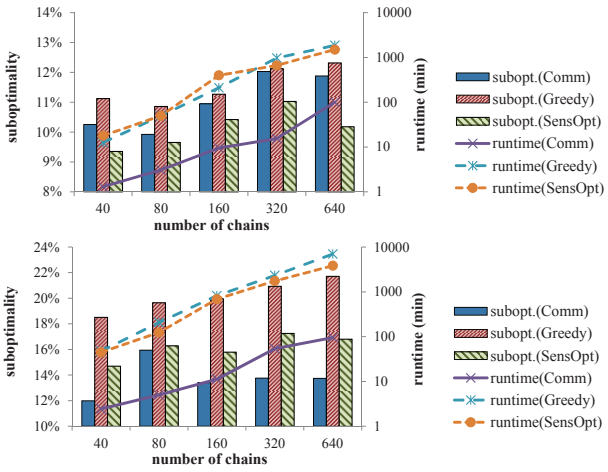


Figure 5: Suboptimality and runtime for different number of chains N (stage K = 40) with EP library (above) and LP library (below).

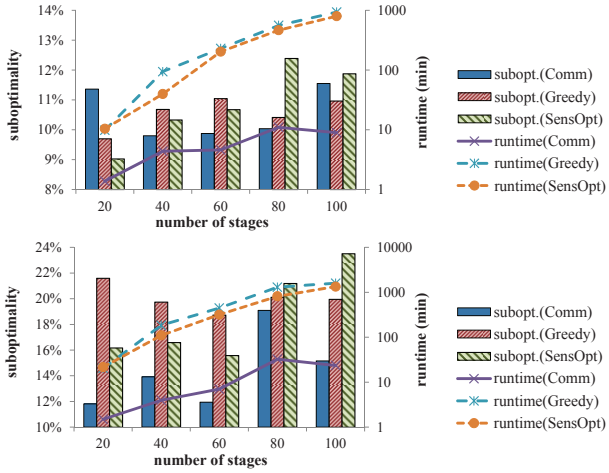


Figure 6: Suboptimality and runtime for different number of stages K (#chain N = 100) with EP library (above) and LP library (below).

timality and runtime increase significantly according to the design complexity. With average net degree of 2.4 and the LP library, large suboptimality ($> 70\%$) is found for each heuristic.

In addition, we study the effect of delay constraints on suboptimality and complexity. Figure 8 shows suboptimality and runtime results when the testcases are generated with different delay constraints. The testcases have the same topology (number of chains, number of stages and net degree). However, the suboptimalities achieved by each heuristic differ widely according to the timing constraint. From the results, netlists with tight delay constraint lead to greater heuristic suboptimality, especially with the LP library.

Table 5 shows suboptimality and runtime results for the generated netlists in Table 4 and one example *eyechart*[7] circuit [15]. The results[8] show that common sizing methods, including two commercial tools (*Comm1*, *Comm2*), are suboptimal for realistic benchmark circuits by up to 16.7%, 52.2%, 29.0% and 26.9% for the commercial tools, greedy method and sensitivity-based method, respectively. Among the testcases, *ng_JPEG* shows the largest suboptimality; we believe that this a consequence of having larger path depth than the other testcases.

Finally, because new realistic benchmarks may not induce the same relative performance across heuristics as real designs, we have also compared the same leakage optimizers using real circuits and real timing/leakage libraries. Table 6 shows the leakage optimization results with the real circuits and libraries. The designs are implemented with a TSMC 65GP library (65nm), and are synthesized, placed and routed with *Synopsys Design Compiler C-2009* [23] and *Cadence Encounter v9.1* [22]. For the leakage optimization, both multi-V_t (NVT, HVT, LVT) library (part (a) of the table) and multi-L_{gate} NVT library (part (b) of the table) have been used. Reported suboptimality is calculated from the best result in the four result columns. In the table, each optimizer shows different suboptimality according to each design. The tools suffer from

[7]Specifically, we use the EP_NLD and LP_NLD *eyechart* circuits with 5ns timing budget from [27].

[8]Results for the *ng_MPEG* testcase are missing for the *UCLA* greedy sizing tool, which cannot handle the large (∼500K instance) instance size.

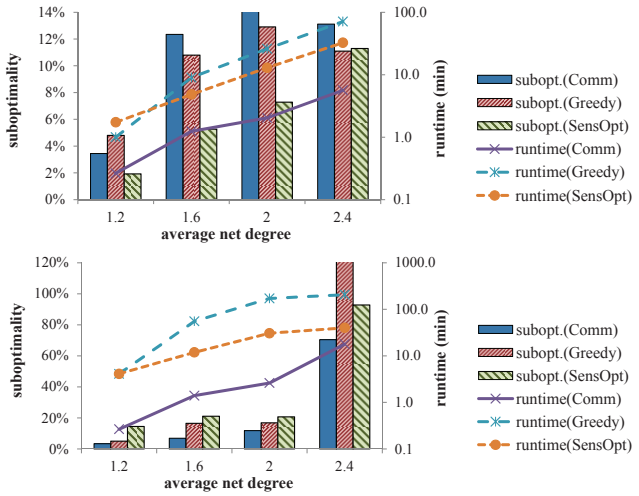


Figure 7: Suboptimality and runtime for different average net degrees (chain N = 40, stage K = 40) with EP library (above) and LP library (below).

significant suboptimality, e.g., *EXU* circuits for *Comm1* (40.8%) and *Greedy* (28.7%), and *AES* circuits for *Comm2* (18.2%). Since suboptimalities are calculated relative to the best heuristic result, there is further suboptimality from the actual optimal solution. It is clear from Tables 5 and 6 that – somewhat unfortunately – the artificial and real netlists and performance libraries suggest different relative and absolute suboptimalities for the sizing heuristics. We believe that this is for several reasons, including (1) our enhanced eyechart-like benchmarks consist of netlists without wire capacitance, and (2) we use only one kind of library cell for each number of fanin ports. We continue to explore ways to improve the matching to results on real designs and real libraries, while maintaining the important property of having a known optimal sizing solution.

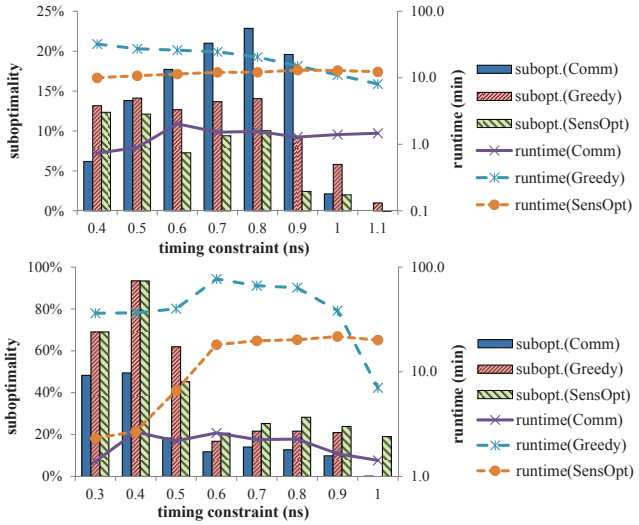


Figure 8: Suboptimality and runtime for different timing constraints (chain N = 40, stage K = 40, average net degree = 2.0) with EP library (above) and LP library (below).

Table 5: Suboptimality with respect to known optimal solution for generated netlists in Table 4.

testcase	optimal leakage (W)	*Comm1*	*Comm2*	*Greedy*	*SensOpt*
(a) EP library					
eyechart	3.46E-6	18.9%	25.4%	24.0%	20.1%
ng_SASC	9.35E-8	10.9%	24.9%	4.9%	2.8%
ng_SPI	2.62E-7	11.4%	21.0%	13.5%	10.9%
ng_AES	5.97E-6	14.2%	13.7%	11.7%	8.8%
ng_EXU	6.62E-6	15.2%	20.9%	10.8%	8.9%
ng_JPEG	3.29E-5	16.7%	33.1%	11.9%	12.0%
ng_MPEG	9.54E-5	10.9%	52.2%	-	10.1%
(b) LP library					
eyechart	3.38E-6	39.3%	44.3%	29.0%	29.1%
ng_SASC	1.55E-7	13.0%	31.4%	22.5%	16.0%
ng_SPI	3.91E-7	6.9%	27.2%	24.3%	24.0%
ng_AES	8.62E-6	11.3%	31.9%	22.4%	23.5%
ng_EXU	9.33E-6	13.9%	40.5%	24.6%	23.4%
ng_JPEG	4.74E-5	16.3%	38.3%	29.0%	26.9%
ng_MPEG	1.65E-4	15.5%	43.7%	-	21.6%

5. CONCLUSIONS

In this work, we have proposed a new benchmark generation technique for gate sizing, which constructs *realistic* circuits with known optimal solutions. Our generated netlists closely resemble real designs in terms of instance count, path depth, interconnect complexity, and net degree / fanin / fanout distributions; all of these attributes are parameters of the netlist generation. When we compare our generated benchmarks with real designs, we also see similarities with respect to other circuit characteristics such as average net degree and Rent parameter.

Our benchmarks with known optimal solutions enable systematic and quantitative study of the suboptimality of common sizing heuristics, with respect to key parameters of the circuit topology. In particular, our experimental results with web-available academic tools and commercial tools show that common leakage-driven sizing methods are suboptimal for realistic benchmark circuits by up to 52.2% and 43.7% for V_t-assignment and CD-biasing formulations, respectively. At the same time, our results also show discrepancies between inferences obtained using our generated circuits and those obtained from real circuits (and libraries). However, all of our results suggest that (1) commercial tools may still suffer from significant suboptimality, and/or (2) existing methods have "similar" degrees of suboptimality, especially as instance size increases (cf. results for JPEG and MPEG in Table 6). Our ongoing work seeks to address the above-mentioned discrepancies. In addition, we are working to handle more realistic delay models, possibly in the context of realistic benchmarks with tight upper bounds on optimal gate leakage.

6. REFERENCES

[1] W. N. Li, "Strongly NP-Hard Discrete Gate Sizing Problems", *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 1993, pp. 468–471.

[2] J. P. Fishburn and A. E. Dunlop, "Tilos: A Polynomial Programming Approach to Transistor Sizing", *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 1985, pp. 326–328.

Table 6: Suboptimality (with respect to best heuristic solution) for real performance libraries and netlists.

testcase	minimum leakage (W)	*Comm1*	*Comm2*	*Greedy*	*SensOpt*
(a) TSMC65 GPLUS Multi-V_t (NVT, HVT, LVT) library					
SASC	2.46E-5	15.3%	5.9%	13.0%	0.0%
SPI	6.26E-5	33.0%	24.3%	10.8%	0.0%
AES	6.29E-4	0.0%	18.2%	13.8%	9.8%
EXU	9.27E-4	17.3%	26.1%	21.0%	0.0%
JPEG	5.76E-3	1.7%	2.7%	3.5%	0.0%
MPEG	1.38E-2	22.3%	18.7%	-	0.0
(b) TSMC65 GPLUS L_{gate} biased NVT library					
SASC	2.85E-5	23.6%	27.7%	4.4%	0.0%
SPI	6.55E-5	1.9%	0.0%	1.4%	0.5%
AES	5.63E-4	30.2%	12.6%	9.1%	0.0%
EXU	9.41E-4	40.8%	3.4%	28.7%	0.0%
JPEG	5.24E-3	13.7%	12.4%	5.7%	0.0%
MPEG	1.81E-2	0.0%	5.3%	-	2.9%

[3] P. Pant, R. K. Roy and A. Chatterjee, "Dual-Threshold Voltage Assignment with Transistor Sizing for Low Power CMOS Circuits", *IEEE Trans. on VLSI Systems* 9(2) (2001), pp. 390–394.

[4] 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* 10(2) (2002), pp. 79–90.

[5] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Gate-length Biasing for Runtime-leakage Control", *IEEE Trans. on Computer-Aided Design* 25(8) (2006), pp. 1475–1485.

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

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

[8] Y. Liu and J. Hu, "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment", *IEEE Trans. on Computer-Aided Design* 29(2) (2010), pp. 223–234.

[9] J. Darnauer and W. W. Dai, "A Method for Generating Random Circuits and Its Application to Routability Measurement", *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 1996, pp. 66–72.

[10] M. D. Hutton, J. Rose, J. P. Grossman and D. Corneil, "Characterization and Parameterized Generation of Synthetic Combinational Circuits", *IEEE Trans. on Computer-Aided Design* 17(10) (1998), pp. 985–996.

[11] D. Stroobandt, P. Verplaetse and J. V. Campenhout, "Generating Synthetic Benchmark Circuits for Evaluating CAD Tools", *IEEE Trans. on Computer-Aided Design* 19(9) (2000), pp. 1011–1022.

[12] L. Hagen, J. H. Huang and A. B. Kahng, "Quantified Suboptimality of VLSI Layout Heuristics", *Proc. ACM/IEEE Design Automation Conference*, 1995, pp. 216–221.

[13] C. Chang, J. Cong and M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms", *Proc. Asia South-Pacific Design Automation Conference*, 2003, pp. 621–627.

[14] J. Cong, M. Romesis and M. Xie, "Optimality and Stability Study of Timing-Driven Placement Algorithms", *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 2003, pp. 472–478.

[15] P. Gupta, A. B. Kahng, A. Kasibhatla and P. Sharma, "Eyecharts: Constructive Benchmarking of Gate Sizing Heuristics", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 597–602.

[16] B. S. Landman and R. L. Russo, "On a Pin versus Block Relationship for Partitions of Logic Graphs", *IEEE Trans. on Computers* C-20(12) (1971), pp. 1469–1479.

[17] C. Chen, C. Chu and D. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation", *IEEE Trans. on Computer-Aided Design* 18(7) (1999), pp. 1014–1025.

[18] K. Jeong, A. B. Kahng and H. Yao, "Rent Parameter Evaluation Using Different Methods", http://vlsicad.ucsd.edu/WLD/RentCon.pdf.

[19] *OpenCores: Open Source IP-Cores*, http://www.opencores.org .

[20] *Sun OpenSPARC Project*, http://www.sun.com/processors/opensparc .

[21] *Blaze MO User's Manual*, http://www.tela-inc.com .

[22] *Cadence Encounter User's Manual*, http://www.cadence.com .

[23] *Synopsys Design Compiler User's Manual*, http://www.synopsys.com .

[24] *Synopsys PrimeTime User's Manual*, http://www.synopsys.com .

[25] *Enhanced OAGear-Static-Timer with Heuristics for Gate Sizing*, http://nanocad.ee.ucla.edu/Main/DownloadForm .

[26] *Sensitivity-Based Leakage Optimizer*, http://vlsicad.ucsd.edu/SIZING/optimizer.html#SensOpt .

[27] *UCLA Eyecharts*, http://nanocad.ee.ucla.edu/Main/DownloadForm .