Discrete Sizing for Leakage Power Optimization in Physical Design: A Comparative Study

SANTIAGO MOK, JOHN LEE, and PUNEET GUPTA, University of California at Los Angeles

While sizing has been studied for over three decades, the absence of a common framework with which to compare methods has made progress difficult to measure. In this article, we compare popular sizing techniques in which gates are chosen from a discrete standard cell library and slew and interconnect effects are accounted for. The difference between sizing methods reduces from roughly 53% to 8% between best and worst case after slew propagation is taken into account. In our benchmarks, no one sizing technique consistently outperforms the others.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Discrete sizing, gate sizing algorithms, leakage power optimization, greedy, linear programming, Lagrangian relaxation, peephole, sensitivity-based sizing

ACM Reference Format:
Mok, S., Lee, J., and Gupta, P. 2012. Discrete sizing for leakage power optimization in physical design: A comparative study. ACM Trans. Des. Autom. Electron. Syst. 18, 1, Article 15 (December 2012), 11 pages.
DOI = 10.1145/2390191.2390206 http://doi.acm.org/10.1145/2390191.2390206

1. INTRODUCTION

Sizing is widely used to tune design parameters (i.e., gate width, Vt) to meet timing, power, and signal integrity constraints. Compared to other optimization methods, such as placement and routing, sizing can help meet these constraints with minimal disturbance on the overall layout.

In library-based designs, the gate sizing problem amounts to choosing an appropriate size from the cell library for each of the gates in the design. This problem has been studied extensively; however, as the problem is NP-hard [Li 1994], optimality and progress have been difficult to show. A variety of approaches have been proposed to the sizing problem.

A rich literature [Fishburn and Dunlop 1985; Wei et al. 1999; Coudert 1997; Sirichotiyakul et al. 2002; Berkelaar and Jess 1990; Srivastava 2003; Nguyen et al. 2003; Chinnery and Keutzer 2005; Liu and Hu 2009; Chen et al. 1999] covers the sizing framework for area and power optimization. Fishburn and Dunlop [1985], Wei et al. [1999], and Coudert [1997] apply a greedy heuristic to optimize the design subject to delay or area constraints. Sirichotiyakul et al. [2002] studes a sensitivity-based heuristic. The sizing problem has been formulated as a linear program (LP) problem [Berkelaar and Jess 1990; Srivastava 2003; Chinnery and Keutzer 2005]. Srivastava [2003] solves the problem as continuous optimization and discretizes the solutions. A

This work was supported in part by the National Science Foundation under grant CCF-0811832. Author's addresses: S. Mok, J. Lee, and P. Gupta, Electrical Engineering Department, University of California at Los Angeles; Corresponding author's email: santiago.mok@gmail.com.
Permission to make digital or hard copies of part or all 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 show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2012 ACM 1084-4309/2012/12-ART15 $15.00
DOI 10.1145/2390191.2390206 http://doi.acm.org/10.1145/2390191.2390206

slack allocation-based linear program is employed in Nguyen et al. [2003]. Lagrangian relaxation (LR)-based optimization is proposed in Liu and Hu [2009] and Chen et al. [1999].

These approaches do not use a common delay model, which makes the comparison between methods hard. For example, a number of works [Fishburn and Dunlop 1985; Wei et al. 1999; Liu and Hu 2009; Chen et al. 1999] approximate timing using Elmore delay. Sirichotiyakul et al. [2002] uses a region-wise quadratic delay model in circuit timing and Elmore delay in estimating sensitivities. Berkelaar and Jess [1990] and Srivastava [2003] assume gate delay as a linear function of gate size or threshold voltage. Nguyen et al. [2003] employ SPICE-characterized rise/fall delay as a function of load.

The difference in the delay model used in the literature is problematic, as it has a large impact on the overall performance of the methods. For instance, a comparison of the slew effects for gate sizing is shown in Section 3, ploting the performance of the two discrete sizing methods (LPS and LR) relative to the performance of a third method (the greedy method). The performance of the LPS and LR methods changes drastically due to slew effects, even under the same timing constraint. On average, the difference between these two methods relative to the greedy method reduces from 53% to 8% when slew propagation is added. We will analyze the rationale for this in Section 3 after we discuss the sizing methods.

Most work on sizing focuses on circuit-level optimization post-synthesis, and the effects in the physical design context have not been carefully addressed. Additional constraints must be made in this context: (1) the discrete gate sizes in a library-based design is the de facto standard; hence, the assumption of continuous sizes cannot be made; (2) interconnect contributes significant capacitive loading, creating a significant impact on the gate delays. The impact of interconnect on the overall design timing cannot be ignored, especially in nanometer designs. However, considerations related to incremental placement and routing (as in Lee and Gupta [2010]) are outside of the scope of this work, along with crosstalk, slew, and other noise violations.

In this article, we will compare popular discrete gate sizing methods implemented on a realistic static timing engine.[1] We also propose a new incremental gate sizing heuristic which attempts to improve results of local sensitivity-based sizing heuristics. The article, is structured as follows. Section 2 discusses commonly employed sizing optimization techniques. We introduce a new incremental sizing framework in Section 2.4. Experimental results are discussed in Section 3, and we conclude our findings in Section 4.

2. DISCRETE GATE SIZING METHODOLOGIES

In this section, we discuss four widely studied gate sizing algorithms: greedy, linear programming-based slack allocation (LPS), linear programming-based assignment (LPA), and Lagrangian relaxation (LR).

2.1. Greedy Algorithm

The greedy algorithm is a heuristic that evaluates the most cost-effective solution at each stage of the optimization. Many gate sizing methods have been centered around the greedy heuristic and sensitivity-based greedy heuristic [Fishburn and Dunlop 1985; Coudert et al. 1996; Coudert 1997; Wei et al. 1999; Sirichotiyakul et al. 1999, 2002].

[1]The UCLA_Timer is based on the open source OAGear timer and can be downloaded from `http://nancoad.ee.ucla.edu`. It supports netlists, libraries, and parasitics in standard formats; it does correct incremental timing analysis, including sequential elements, and it accounts for interconnect delay and slew propagation.

Table I. Greedy Method Comparisons

	(1) Entire Fanout		(2) Neighborhood		(3) None	
	%Impr	time(s)	%Impr	time(s)	%Impr	time(s)
c1355	40.62%	362	38.33%	22	36.25%	25
c1908	38.35%	308	38.80%	15	36.94%	18
c5315	19.88%	2057	19.18%	36	18.38%	33

Note: The (1) entire fanout cone is updated after each gate change. (2) A neighborhood of the immediate fan-ins and fan-outs of a changed gate are updated; (3) Sensitivities are updated only after all the sensitivities are evaluated.

We implemented a greedy heuristic that optimizes leakage power based on a ΔPower/ΔDelay sensitivity function (as in Fishburn and Dunlop [1985]) that measures the power improvement per unit delay. As in Coudert [1997] and Gupta et al. [2005], the algorithm starts with a design that has substantial slack and trades gate delay for leakage power in order of decreasing sensitivity values. For each gate size change, an incremental static timing analysis is carried out to ensure that timing is met. This heuristic iterates until no further gate size changes are possible that would meet timing and improve power. The flow of the algorithm is as follows.

(1) *Create Sensitivity List.* For each gate in the circuit, compute the sensitivity value ΔPower/ΔDelay with respect to each of the available library cells that are smaller than the current one.
(2) *Evaluate.* For each sensitivity (in descending order), change the current gate to the new gate's size. Perform incremental timing analysis and verify timing feasibility. If it does not meet timing constraints, revert the gate to its previous size.
(3) *Update and Iterate.* After any sizing modification in the circuit, sensitivity values are updated for the immediate fan-ins and fan-outs of the modified gate. While the entire fan-out cone may be affected, only the immediate fan-ins and fan-outs are updated, as the performance improvements are small (see Table I for a comparison of the update method vs. the improvements). The evaluate phase loops through all power-improving moves until the list is exhausted. If any sizing changes were performed, the sensitivity list is updated to account for the new configuration. Otherwise, the heuristic terminates, as no further improvement is possible.

2.2. Linear Programming Slack Allocation and Assignment

The standard linear programming (LP) problem minimizes a linear function subject to linear equality and inequality constraints. LP has been widely studied for gate sizing and Vth assignment [Berkelaar and Jess 1990; Nguyen et al. 2003; Srivastava 2003; Chinnery and Keutzer 2005; Jeong et al. 2009].

The gate sizing technique in Nguyen et al. [2003] uses LP-based slack allocation (LPS). Similar to our greedy approach, a (ΔPower/ΔDelay) sensitivity value is assigned for each library size available from each gate in the circuit. Based on the power sensitivity value, the slack is allocated to each individual gate to minimize leakage power. It has the additional capability to consider the topology of the design and allocates more delay to those gates that are bottlenecks and common to multiple critical paths. In this comparative study, we implemented the slack allocation-based LP algorithm similar to that of Nguyen et al. [2003].

The technique in Chinnery and Keutzer [2005] uses LP-based assigment (LPA) to assign a library cell to a gate directly. In this formulation, each variable represents a decision as to whether a gate is assigned to a particular library cell. After the linear program is solved, if the assigment variable is ≈ 1, the gate is assigned; the gate is not assigned otherwise.

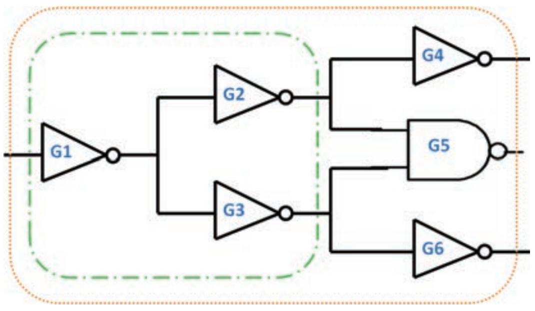

Fig. 1. Peephole example

2.3. Lagrangian Relaxation

Lagrangian relaxation is another mathematical programming-based method that has been adapted to solve the sizing problem [Chen et al. 1999; Liu and Hu 2009]. In this study, we implemented the Timing-Constrained Power Optimization in Liu and Hu [2009], which is a dynamic programming (DP)-guided Lagrangian relaxation heuristic.

This algorithm offers two main advantages over continuous Lagrangian-based optimizations. (1) It is easily adapted to a lookup table timing model, which avoids inaccurate data fitting to a continuous timing model. (2) It does not need to snap the solution to a discrete size, which is a process that is prone to error.

In brief, the algorithm works by converting the timing-constrained power optimization problem into an unconstrained problem using Lagrange multipliers. This results in two subproblems: (1) the problem of minimizing the weighted sum of delay and power in Eq. (1),

$$\begin{array}{lll} \text{minimize}_x & \sum_{x\,\in\,G} p(x) + \sum_j \sum_{i\,\in\,\text{fanin}(j)} \lambda_{ij} D(x_j) & \\ \text{subject to} & X_{\min} \le x_n \le X_{\max}, & \forall n \end{array} \tag{1}$$

and (2) the Lagrangian dual problem of updating the Lagrange multipliers (λ) to maximize the Lagrangian function.

2.4. Peephole

In this section, we present an incremental optimization method that can be used after and on top of any other sizing heuristic. This method focuses on optimizing portions of the design, which we call *peepholes*, which can further reduce the leakage power of an already optimized design.

Definition 1. A *peephole* is a collection of a given gate (*the root*) and an arbitrary collection of its fan-outs.

In this work we focus on a particular type of peephole.

Definition 2. An n^{th} degree peephole consists of the peephole formed by the root and all fan-outs that have a minimum distance of at least n levels from the root.

Note that the n^{th} degree peephole excludes gates that are less than n levels from the root. For example, in Figure 1, the 2^{nd} degree peephole is $\mathcal{P} = \{G1, G4, G5, G6\}$ and excludes gates $\mathcal{P} = \{G1, G2, G3\}$.

Peephole optimization focuses on upsizing the root gate to create slack, which is used downstream to downsize multiple gates. This improves leakage power by downsizing multiple gates at the expense of increasing the leakage power of the single root gate,

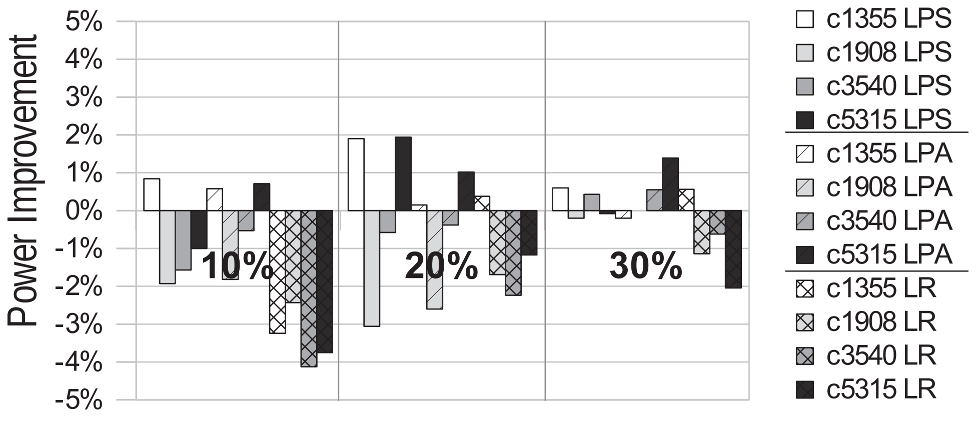

Fig. 2. Power improvement vs. delay constraint. Delay constraints of 10%, 20%, and 30% of the spread between minimum and maximum delay are shown.

yielding a net leakage power improvement. This allows improvements in cases where it is not possible by modifying only a single gate.

2.4.1. Peephole Optimization Algorithm. In the peephole algorithm, all peepholes for a given circuit are stored in a list L. Unnecessary peepholes are pruned before evaluation, and the method evaluates the remaining peepholes. The method relies on two main routines, pruning and evaluation.

—*Prune (Peephole Selection)*. The total number of peepholes (of all degrees) in a circuit is prohibitively large, and (1) pruning is necessary to manage runtime, as no significant improvement is obtained with fan-outs higher than 3rd degree; (2) peepholes with root nodes at maximum size are removed; and (3) peepholes with roots that have all fan-outs at minimum size are also removed.

—*Evaluate*. Check if a peephole $\mathcal{P}(r, j)$ yields leakage power improvement without violating timing. In a peephole, the combination with the largest net power improvement is chosen.

After peephole optimization ends, excessive slack is leftover. The sensitivity-based greedy heuristic is applied after peephole optimization to take advantage of the slacks.

3. EXPERIMENTAL SETUP AND RESULTS

All heuristics are implemented in C++ using the OpenAccess API[2] and run on a server with eight Intel® Xeon® processors running at 2GHz with 18GB of RAM. We used the UCLA_Timer [Mok 2011] (based on OAGEAR) for static timing analysis (STA) and to verify the timing and power of the resulting designs. We tested the algorithms on ISCAS'85 [Hansen et al. 1999] and ITC'99 [Corno et al. 2000] benchmark circuits. All benchmark circuits are synthesized with a Cadence RTL Compiler.

The designs were optimized for leakage power and timing during synthesis with tightest delay target. We used Cadence SoC Encounter vb.2[3] to place, route, and extract interconnect RC (.spef). The placed and routed benchmark circuits were then timing optimized using four optimization methods which were implemented on top of the UCLA_Timer.

[2]OpenAccess OAGear and Nangate. http://www.si2.org.
[3]http://cadence.com.

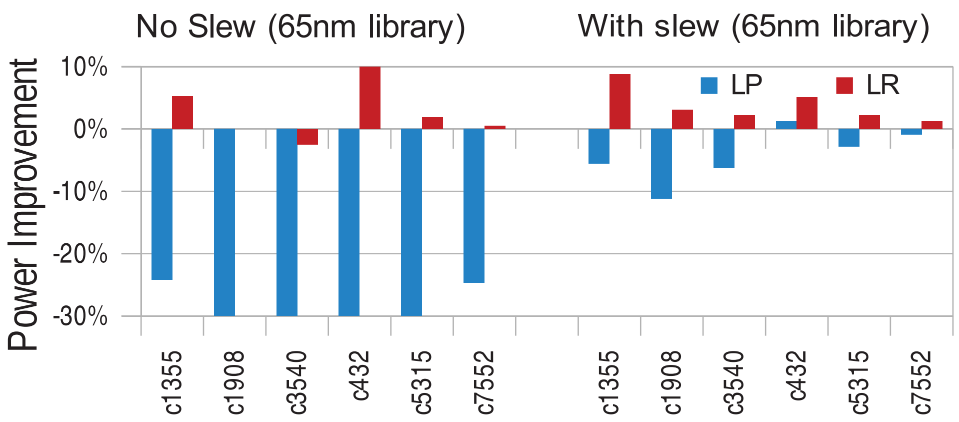

Fig. 3. Comparing optimization with and without slew.

The target timing constraint is obtained at 30% between minimum delay and all gates with minimum size configuration. As the delay spread between minimum delay and minimum size is quite small (<200ps for 500ps designs), 30% is the right target delay to emphasize the difference between methods[4]. The benchmark circuits were timing optimized using the Slack Minimization DP heuristic [Liu and Hu 2009] implemented with the UCLA_Timer.

The benchmark circuits are synthesized to a Nangate 45nm and a commercial 65nm library (comm-lib). In the Nangate library, there are six sizes for inverters and buffers and three sizes for other standard cells. In the comm-lib, there are at least six sizes for all standard cells. In addition to gate sizing, we synthesized the circuits to different Vt (high, standard, and low) from the comm-lib to evaluate Vt assignment.

Slew effects have a significant impact on gate sizing for the LPS algorithm, as shown in Figure 3, where the results for the LPS and LR algorithms are shown relative to the performance of the greedy method. The results without slew correlate well with the results comparing LR and LPS in Liu and Hu [2009], which also does not account for slew. Though LPS is used to smartly distribute slacks, the allocation is not exact due to rough estimates of delay and sensitivities. Without slew effects, the initial slack allocation dominated the amount of power improvement possible, rendering no slack left for further iterations. On the other hand, when slew is propagated, LPS benefits from having multiple iterations to allocate slacks, and the difference between the methods is smaller.

The gate sizing results for the algorithms are plotted in Figure 4 for Nangate and comm-lib. The sizing results are plotted with respect to greedy heuristic results. Higher bars indicate a larger power improvement over greedy.

The results show that there is no algorithm that consistently provides good results. The performance varies between the ISCAS '85 benchmarks and the ITC'99 benchmarks and also between the libraries. For the Nangate library, the methods perform similarly for the ISCAS '85 benchmarks, and the greedy method fares the best for the ITC'99 benchmarks—approximately 16% better. For the commercial 65nm library with gate sizing, the LR method performs the best on average for the ISCAS '85 benchmarks, with an average improvement of 3.2% over greedy; in the case of the ITC '99 benchmarks, the methods all perform similarly. For the commercial 65nm library with

[4]Figure 2 shows how the delay constraint affects the performance. In these cases, the sizing heuristics perform worse on tighter timing constraints.

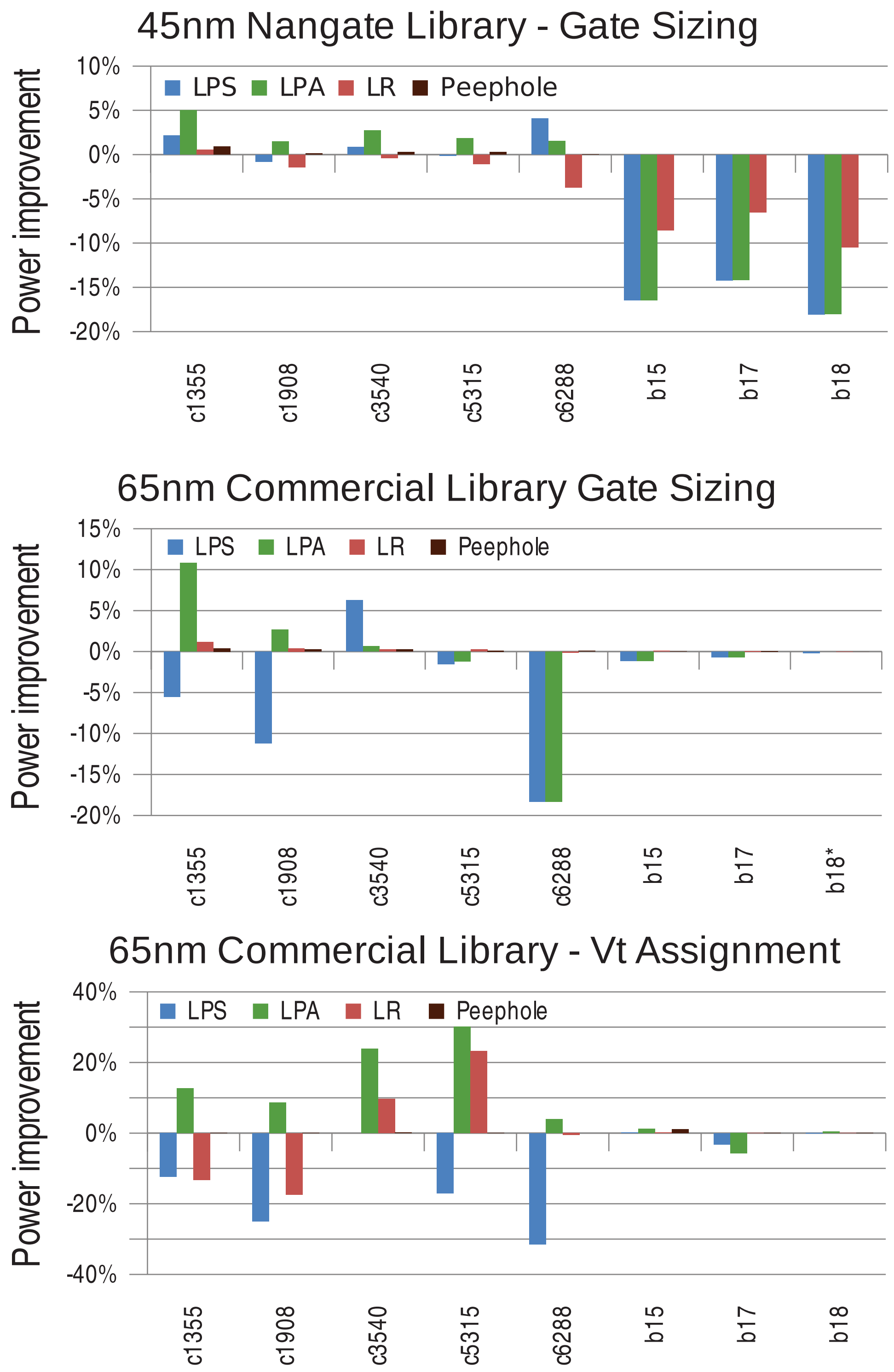

Fig. 4. Gate Sizing results. *Note that the b18 LPA result for the ST library is omitted in gate sizing.

Vt assignment, the LPA method performed the best in the ISCAS '85 benchmarks, with an average improvement of 15.8%. Note that the Vt assignment provides a larger range in powers due to the exponential dependence of leakage power on the threshold voltage. In the corresponding ITC'99 benchmarks, the results were similar between all methods.

With regard to the Nangate gate sizing library, the resulting power is very close among all the algorithms except for ITC'99 benchmarks. In contrast, the comm-lib with gate sizing has significant differences for the ISCAS'85 benchmarks, but insignificant differences for the ITC'99 benchmarks. One explanation for this discrepancy is the small gate size set in the Nangate library. The difference in power results among the algorithms is much more pronounced in the Vt assignment library (between -9.1% to +13%, on average) than in the gate sizing libraries (between -3% to -0.6%, on average). LPA performed the best in many of the benchmarks for gate set in the library with exponential differences. On the other hand, the LR algorithm requires significant tuning of the Langrangian multipliers in order to optimize the results, but there is not a single sweet spot that would optimize all the types of libraries we tested. The peephole method was only able to add 1%–5% on top of what greedy had optimized.

The runtime comparison is shown in Table II. Greedy and LR report similar runtimes. In a few cases, the greedy runtime lags behind LR on the ITC'99 benchmark. Greedy takes 40+ iterations to converge, while the LR reaches its target timing within 30 iterations, and the LPA is fixed to run for 20 iterations. The LPS and LPA algorithms have the worst runtimes among the four, which is due solely to its need to use many more calls to the timer. In LPS and LPA, the slack allocation is quite sensitive to the rough estimate of the delta delay value; hence, static timing analysis (STA) is carried out for the sensitivity computation. Also, after slack allocation, STA is triggered to search for the library gate assignment that fits the given allocated delay. However, triggering the STA was not necessary in greedy and LR, and the delay calculations can be approximated, resulting in significant speedups. The LPA algorithm is the slowest, and the b18 benchmark for the commercial 65nm library could not be run due to excessive runtimes.

Note that the implementations are not optimized for runtime and that optimization may improve these numbers. For example, a convergence plot is shown in Figure 5. Note that the greedy, LPA, and LPS methods start with timing-feasible designs, while the LR starts with a timing-infeasible design. The tail and the stopping criterion can be optimized to reduce runtime.

In terms of calls to the STA, the LPS, LPA, and LR algorithms have a complexity of $O(N \cdot m)$ per iteration, and the greedy method has a complexity of $O(N \cdot m \cdot q)$ per iteration, where N is the number of gates, m is the maximum number of gate sizes (or Vt options), and q is the maximum number of fanouts.[5] These expressions explain why the runtimes for the commercial library sizing are greater than those of the Nangate and the Vt assignments (the number of options is much greater). However, it does not explain the runtime numbers between methods; most of these differences are due to factors not captured in complexity analysis.

4. CONCLUSION

Four distinct discrete sizing methods were compared using a common timer that employs table-based delay calculation. In this study, we find that certain algorithms do favor certain types of library sizing options. For instance, the LPA algorithm has much better results in Vt assigment compared to gate sizing. Likewise, the LR algorithm had much better results on average on the gate sizing benchmarks. On the larger ITC'99

[5]The number of iterations is insensitive to the number of gates.

Table II. Runtime Comparison

	Nangate (45nm)						**comm-lib (65nm)**						**Vt Assignment comm-lib (65nm)**					
	No. of	Runtime (s)					No. of	Runtime (s)					No. of	Runtime (s)				
design	Gates	Greedy	LPS	LPA	LR	Peep	Gates	Greedy	LPS	LPA	LR	Peep	Gates	Greedy	LPS	LPA	LR	Peep
c1355	601	18	241	410	33	62	411	62	441	3166	69	196	572	19	245	147	68	62
c1908	549	12	221	327	20	53	478	104	691	5048	106	346	577	23	305	336	82	53
c3540	1086	43	931	1185	98	156	1006	273	4093	15998	157	917	1377	76	2535	2364	159	156
c5315	1237	28	560	839	143	87	1094	157	1971	10729	188	383	1684	53	1760	2794	119	87
c6288	3342	439	8622	11572	997	572	2875	2117	32861	89123	2196	3413	4073	569	24954	13301	910	730
b15	7833	383	11274	7486	269	6	7099	8946	93527	430771	5879	525	9965	1075	46537	73848	1026	157
b17	26555	2068	74705	201718	3082	24	23100	32069	246377	1119453	21043	987	30631	4755	253239	538592	7749	188
b18	64584	5856	174882	579780	23072	8	61537	192955	1044031	*	28688	4277	79547	29680	72331	218045	48377	798

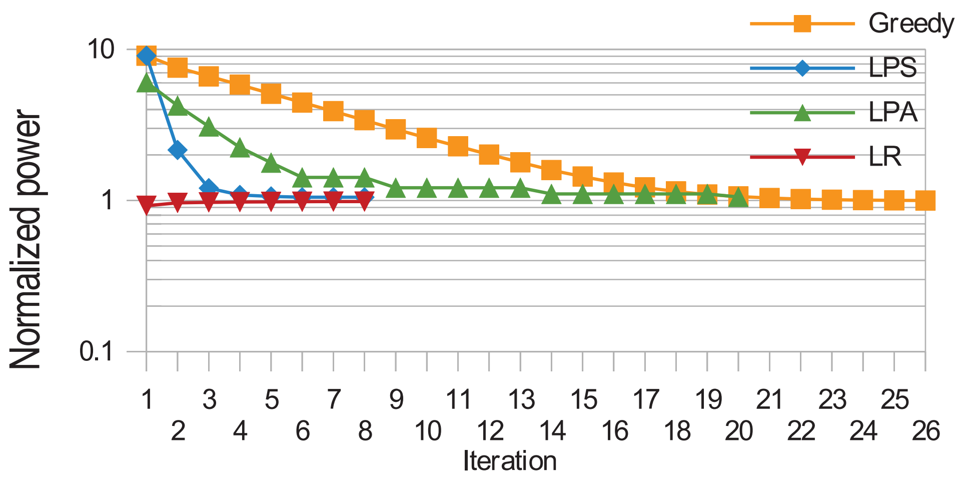

Fig. 5. Power vs iteration for the greedy, LPS, and LR heuristics on the c7552 benchmark with 65nm comm-lib

benchmarks with the small set of gate sizes on Nangate Lib, the LPS, LPA, and LR did not perform as well as greedy. We find that no algorithm consistently leads to better results in general. In our implementations, greedy sizing was much faster on average than other methods, which indicates that sensitivity-based sizing may be a good choice for most applications.

Though we have not shown significant improvement with our peephole method, incremental gate sizing has not been studied in the literature, and it opens many interesting areas of research for late-stage optimization.

ACKNOWLEDGMENTS

We would like to acknowledge discussions with Dr. Andrew Kahng (UCSD) and Dr. Saumil Shah (Magma). Greedy sizing implementations, along with the UCLA_Timer, are available at http://nanocad.ee.ucla.edu/Main/DownloadForm.

REFERENCES

Berkelaar, M. R. C. M. and Jess, J. A. G. 1990. Gate sizing in mos digital circuits with linear programming. In *Proceedings of the International Conference on Europeon Design Automation (EURO-DAC'90)*. 217–221.

Chen, C., Chu, C., and Wong, D. 1999. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. *IEEE Trans. CAD 18,* 7, 1014–1025.

Chinnery, D. G. and Keutzer, K. 2005. Linear programming for sizing, vth and vdd assignment. In *Proceedings of the International Conference on Low Power Electronics and Design*. 149–154.

Corno, F., Reorda, M., and Squillero, G. 2000. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Trans. Des. Test Comput. 17,* 3, 44–53.

Coudert, O. 1997. Gate sizing for constrained delay/power/area optimization. *IEEE Trans. VLSI Syst.*

Coudert, O., Haddad, R., and Manne, S. 1996. New algorithms for gate sizing: A comparative study. In *Proceedings of the Design Automation Conference 33*, 734–739.

Fishburn, J. and Dunlop, A. 1985. TILOS: A posynomial approach to transistor sizing. In *Proceedings of the International Conference on Computer-Aided Design*.

Gupta, P., Kahng, A. B., and Sharma, P. 2005. A practical transistor-level threshold voltage assignment methodology. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED'99)*.

Hansen, M., Yalcin, H., and Hayes, J. 1999. Unveiling the ISCAS-85 Benchmarks: A case study in reverse engineering. *IEEE Des. Test 16.* 3, 72–80.

JEONG, K., KAHNG, A., AND YAO, H. 2009. Revisiting the linear programming framework for leakage power vs performance optimization. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED'99)*. 127–134.

LEE, J. AND GUPTA, P. 2010. Incremental gate sizing for late process changes. In *Proceedings of the International Conference on Computer Design*. 215–221.

LI, W. 1994. Strongly np-hard discrete gate-sizing problems. *IEEE Trans. CAD 13.* 8, 1045–1051.

LIU, Y. AND HU, J. 2009. A new algorithm for simultaneous gate sizing and threshold voltage assignment. In *Proceedings of the International Symposium on Physical Design (ISPD'09)*. 27–34.

MOK, S. 2011. Propagation delay approximation considering effective capacitance and slew degradation. Tech. rep., UCLA. http://nanocad.ee.ucla.edu/pub/Main/Publications/MSTR4_paper.pdf.

NGUYEN, D., DAVARE, A., ORSHANSKY, M., CHINNERY, D., THOMPSON, B., AND KEUTZER, K. 2003. Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization. In *Proceedings of the International Conference on Low Power Electronics and Design*. 158–163.

SIRICHOTIYAKUL, S., EDWARDS, T., OH, C., PANDA, R., AND BLAAUW, D. 2002. Duet: An accurate leakage estimation and optimization tool for dual-Vt circuits. *IEEE Trans. VLSI Syst.* 70–90.

SIRICHOTIYAKUL, S., EDWARDS, T., OH, C., ZUO, J., DHARCHOUDHURY, A., PANDA, R., AND BLAAUW, D. 1999. Stand-by power minimization through simultaneous threshold voltage selection and circuit sizing. In *Proceedings of the Design Automation Conference*. 436–441.

SRIVASTAVA, A. 2003. Simultaneous vt selection and assignment for leakage optimization. In *Proceedings of the International Conference on Low Power Electronics and Design*. 146–151.

WEI, L., CHEN, Z., ROY, K., AND DE, V. 1999. Design and optimization of dual threshold circuits for low voltage low power applications. *IEEE Trans. VLSI Syst.*. 16–24.

Received June 2011; revised December 2011, June 2012; accepted September 2012