Open access • Proceedings Article • DOI:10.1145/2429384.2429428

Sensitivity-guided metaheuristics for accurate discrete gate sizing — Source link

Jin Hu, Andrew B. Kahng, Seokhyeong Kang, Myung-Chul Kim ...+1 more authors

Institutions: University of Michigan, University of California, San Diego

Published on: 05 Nov 2012 - International Conference on Computer Aided Design

Topics: Metaheuristic and Power optimization

Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing

Jin Hu[†], Andrew B. Kahng[‡+], SeokHyeong Kang[‡], Myung-Chul Kim[†] and Igor L. Markov[†]
[†]University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109, {jinhu, mckima, imarkov}@eecs.umich.edu
UC San Diego, [‡]ECE and [+]CSE Depts., La Jolla, CA 92093, {abk, shk046}@ucsd.edu

ABSTRACT

The well-studied gate-sizing optimization is a major contributor to IC power-performance tradeoffs. Viable optimizers must accurately model circuit timing, satisfy a variety of constraints, scale to large circuits, and effectively utilize a large (but finite) number of possible gate configurations, including V_t and L_g. Within the research-oriented infrastructure used in the ISPD 2012 Gate Sizing Contest, we develop a metaheuristic approach to gate sizing that integrates timing and power optimization, and handles several types of constraints. Our solutions are evaluated using a rigorous protocol that computes circuit delay with Synopsys PrimeTime. Our implementation Trident outperforms the best-reported results on all but one of the ISPD 2012 benchmarks. Compared to the 2012 contest winner, we further reduce leakage power by an average of 43%.

1. INTRODUCTION

The sizing problem in VLSI design seeks to determine design parameters (e.g., gate width and threshold voltage) for each gate, so as to optimize timing, area and power of a circuit, subject to constraints. The problem has been extensively studied, and a number of heuristics have been proposed. However, there have been no definitive comparisons (empirical or mathematical) of different techniques. Moreover, many published techniques make unrealistic assumptions about the underlying circuits, such as the possibility of *continuous gate sizing and V_t assignment* and the convexity of delay functions. Some publications neglect the effect of rounding when using a discrete gate library, or do not account for realistic capacitance and slew constraints. Scalability to circuits with hundreds of thousands of gates is also an important issue, whereas many previous publications use much smaller benchmarks mapped into outdated technologies. To address these shortcomings in published literature, Intel researchers have recently prepared and released an extensive infrastructure for research on large-scale gate sizing [23]. This includes (i) a set of benchmarks ranging from small to large, mapped into a modern discrete gate library, and (ii) a set of evaluation protocols that includes checking timing constraints using industry-standard software (*Synopsys PrimeTime* [33]) and measuring total leakage power for a particular sizing solution. This infrastructure has been used in the ISPD 2012 Gate Sizing Contest which provided a definitive baseline for further research on this topic.

Our research reported in this paper focuses on large-scale optimization of gate sizes under realistic circumstances. Our techniques accurately track circuit timing throughout the optimization process, ensure the satisfaction of several types of constraints, and identify the gates with the greatest impact on power-performance tradeoffs. Rather than approximate gate sizing by continuous convex optimization, as is done in many prior publications, we fully account for the discrete nature of the problem and the nonconvexities of circuit delay functions. Our optimizations try not to overlook available opportunities to improve power-performance tradeoffs, but are also fast and can quickly traverse large regions of the solution space. They are highly modular, and are organized in a hierarchy where high-level metaheuristics configure and drive heuristics, which are assembled from lower-level optimization blocks. The lower-level optimizations are in turn based on high-performance timing and power analysis, constraint repair, search, gain calculations, sorting and prioritization, as well as roll-back. Some of these ideas have been known before, but every hierarchical level in our techniques contains some novel elements. Moreover, in a highly studied and competitive field such as gate sizing, reliable individual components form only a small fraction of overall success — a larger fraction of success is in the selection, ordering and composition of these components, as well as the overall flow. Our most important decisions rely on new insights into large-scale gate sizing and its interaction with design constraints. The main contributions of our work are summarized below.

- We use a sensitivity-guided metaheuristic approach based on *sequential importance sampling* [1] that integrates power and timing optimization, and handles several types of constraints.
- We propose to use Total Negative Slack (TNS) rather than just the worst slack within sensitivity functions, but observe that the impact of individual gate sizing changes on TNS takes too long to compute. Therefore, we apply a fast technique to estimate TNS impact of gate sizing.
- We define a parameterized space of sensitivity functions for gate sizing and traverse this space using a multistart technique that naturally lends itself to efficient parallelization on multi-core and shared memory CPU architectures, and distributed systems.
- Empirical results on the ISPD 2012 contest benchmarks show that our heuristics further improve the best-found solutions in the contest on 13 out of 14 benchmarks by 11% (Table 3). Given that different contestants excelled on different benchmarks, our sizer Trident outperforms any one contestant by a wide margin.

The rest of this paper is organized as follows. Section 2 covers background and reviews prior work. Section 3 introduces our gate sizing heuristics and their details. Section 4 provides experimental results and analysis. Section 5 concludes the paper.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA

2. BACKGROUND AND PRIOR WORK

The problem of sizing standard cells in digital circuits has been extensively studied due to its importance, and many optimization techniques have been developed. Before discussing specific ideas, we introduce the optimization objectives and relevant constraints.

2.1 Gate Sizing Formulation

Consider a netlist $\mathcal{N} = (\mathcal{C}, I, \mathcal{P})$, where $\mathcal{C}$ is the set of cells, I is the set of interconnects between cells, and $\mathcal{P}$ is the set of pins on $\mathcal{C}$. Also given is the input technology library $\mathcal{L}$, where each cell $c \in \mathcal{C}$ has a set of valid (manufacturable) sizes. Given $\mathcal{N}$ and $\mathcal{L}$, the objective of (discrete) gate sizing is to map $\mathcal{C}$ to $\mathcal{L}$ while minimizing the total power consumption subject to design constraints. We consider (*i*) slack constraints, (*ii*) capacitance constraints, and (*iii*) slew constraints.

2.2 Previous Work on Gate Sizing

The vast majority of gate sizing research has focused on finding *continuous* solutions, where parameters can take values within certain contiguous ranges. Some techniques optimize transistor parameters (e.g., threshold voltage), and others focus on entire standard cells and deal with drive strength and input-pin capacitances. However, as modern digital methodologies use only discrete cell types, continuous-valued parameters must be efficiently rounded to allowed discrete values.

Continuous methods. As the gate-sizing problem is easily formulated as an optimization problem, many different implementations have shown success, including:

- Linear programming [2, 5, 17] and network flow [25]
- Convex nonlinear optimization [6, 26, 27, 31], including Lagrangian relaxation [4, 6, 19, 30, 31]
- Slew budgeting [14]

Due to the complexity of modern designs, the two most scalable approaches are those based on *Lagrangian relaxation* and *sensitivity*. Instead of satisfying every imposed constraint, Lagrangian relaxation changes the original constrained problem into an unconstrained problem such that the solutions to the latter can be mapped back to the former. However, Lagrangian relaxation is typically formulated for continuous-variable functions and does not naturally handle discrete gate sizes. Numerical optimization techniques used with Lagrangian relaxation sometimes also assume convexity, which does not hold for practical circuit-delay models.

Discrete methods. In contrast to continuous methods, discrete methods assume a fixed cell (and technology) library with a set of discrete cell sizes. While discrete methods avoid the difficult "rounding" problem that continuous methods face, they must directly solve an NP-hard problem [18]. Branch-and-bound [24] and dynamic programming [15, 19, 22] based approaches can be categorized as discrete.

Another discrete method, the sensitivity-based (iterative) approach, modifies individual components one at a time, such as by changing (*i*) transistor width, (*ii*) threshold voltage (in dual-V_t designs), (*iii*) source voltage (in dual-V_{DD} designs), and (*iv*) gate and/or wire resistances and capacitances. To further improve performance, the authors of [7] developed a multi-directional search, and the authors of [3] suggested using multistarts.

Recently, the authors of [22] developed a Lagrangian-relaxation graph-based approach to efficiently assign cell types in very large netlists. Their optimizations have significantly improved power-performance tradeoffs in ICs designed at Intel. This work has also spurred the ISPD 2012 (Discrete) Gate Sizing Contest [23].

2.3 The ISPD 2012 Gate Sizing Contest

To more accurately model the discrete gate-sizing problem for modern technologies, researchers from Intel organized the ISPD 2012 Gate Sizing Contest [23]. Here, contestants are given (*i*) a set of benchmarks and (*ii*) a (simplified) modern standard-cell library. The benchmarks are derived from the IWLS 2005 benchmarks [16] and have between 25K and 995K cells. The contest employs standard industry formats for benchmarks, including Verilog netlists, interconnect parasitics in IEEE SPEF format, and timing constraints in Synopsys Design Constraints (SDC) format. The library implements 12 different logic functions, and 30 different cell types (options) for each logic function — three different threshold voltages (V_t) and ten different sizes for each V_t. The cell library also has lookup tables for (*a*) delay and transition time (slew), and (*b*) max load capacitance (in the Synopsys Liberty format) for each cell type. An industry timing engine – Synopsys PrimeTime – is used as the reference timer. For simplicity, the contest does not capture false paths, placement-dependent distributed interconnect parasitic models, or multiple clock domains. The contest compares leakage power of violation-free solutions. Specific constraints include (i) zero negative slack on all pins, (ii) a $300ps$ transition time upper bound, and (iii) maximum load capacitance per cell type.

2.4 Stochastic Combinatorial Optimization

High-dimensional, hard combinatorial optimization problems do not admit such mathematically elegant solutions as Lagrangian relaxation, and are often solved using simulated annealing or other metaheuristics that combine local search with a global strategy [9] (e.g., *stochastic hill-climbing*, *tabu search*, or *genetic crossover*). These techniques are difficult to implement well (e.g., require heavy parameter tuning), are generally not reproducible, and require sophisticated mathematics to analyze their asymptotic performance.

The *go-with-the-winners (GWTW)* metaheuristic [1] repeatedly invokes greedy heuristics within randomized multistarts to (i) explore a large search space by maintaining a small set of best-seen solutions, and (ii) find a global optimum with high probability, as proven in [1]. Local minima are avoided when better intermediate solutions are found (e.g., in parallel search threads) and yield multiple derived solutions. GWTW is asymptotically more efficient than a single round of independent randomized restarts, and is on par with simulated annealing but significantly easier to analyze [1].

The maintenance of the best-seen solutions in GWTW is like the "survival of the fittest" invariant in genetic algorithms [9]. However, GWTW does not employ genetic crossover, does not exclude repeat solutions, and allows the number of kept solutions to vary. GWTW can be viewed from the statistics perspective as *sequential importance sampling* and traces its roots to statistical physics [10].

3. OUR METAHEURISTICS

Our proposed heuristic has two stages – Global Timing Recovery (*GTR*) and Power Reduction with Feasible Timing (*PRFT*). *GTR* first seeks violation-free (feasible) solutions, and then *PRFT* iteratively reduces total leakage power of sizing solutions by local search, as illustrated in Figure 1. At each stage of our optimization flow, we parameterize the space of sensitivity functions, and traverse this space to find the best configurations of sensitivity by independent multistarts (Figure 2). After each multistart, we compare all obtained solutions and retain the best/non-dominated solutions. This is accomplished by adapting the go-with-the-winners (GWTW) metaheuristic (Section 2.4). However, our optimization is purely deterministic in that our multistart procedure begins with the small set of the best-seen solutions, whereas GWTW is typically randomized. Solutions after each stage are ensured to be feasible, which enables pruning of dominated solutions by GWTW.

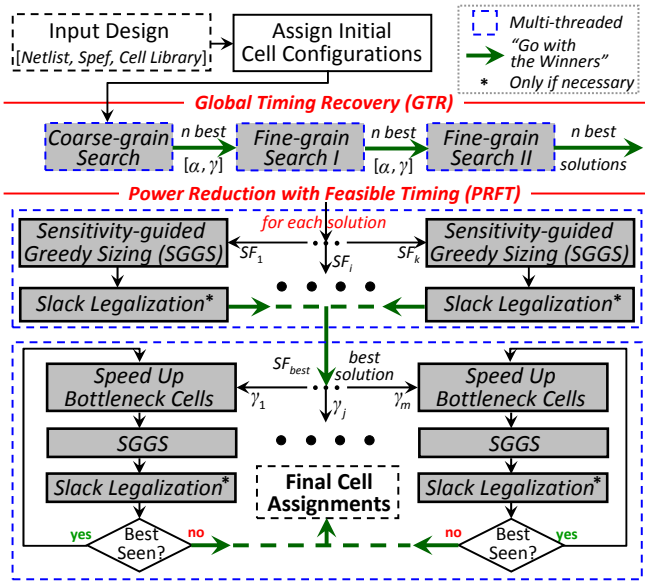

Figure 1: During GTR (Section 3.1), our algorithm maintains the n best-seen timing-valid solutions by performing one stage of coarse-grain (global) search followed by two stages of fine-grain (local) search. Then, during PRFT (Section 3.2), we recover power while respecting timing constraints.

3.1 Global Timing Recovery

This stage starts with an arbitrary cell configuration that is incrementally refined by increasing/decreasing gate sizes or downscaling/upscaling threshold voltages. The interaction of these steps in our implementation has been optimized for the case where the initial solution generally underestimates optimal power dissipation of individual gates and likely violates timing constraints. Therefore, we start with timing recovery by upsizing gates or downscaling threshold voltages. We observe that best-seen solutions for several ISPD 2012 benchmarks configure most cells at or close to their minimum-leakage configurations (Table 3), making minimum-leakage configurations for all cells an appropriate initial setting. For benchmarks with tighter timing constraints, alternative initial settings may save runtime. However, our optimization techniques

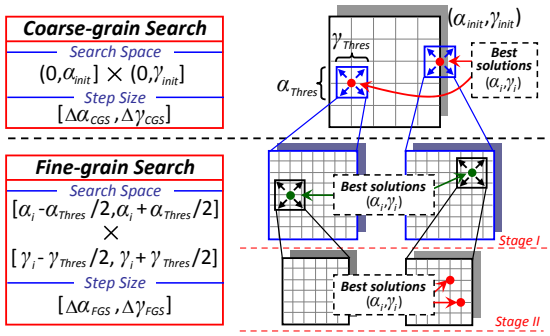

Figure 2: Search-range changes during the GTR search procedure (Algorithm 1). During coarse-grain search, the algorithm sweeps parameters α and γ. Then, fine-grain search local search focuses on ranges around the best-seen (α,γ) from coarse-grain search. Compared to each previous search stage, the ranges (as well as the step sizes) of both α and γ are reduced. After all search stages, GTR produces a set of the best-found solutions based on leakage power.

Algorithm 1 Global Timing Recovery (GTR).

Procedure $TimingRecovery(N)$
Inputs : $power\_exponent$ $0 \leq \alpha \leq 3.0$, $commit\_ratio$ $0 < \gamma \leq 60\%$
Output : sizing solution S

1. Set the current solution S with a minimum-leakage setting;
2. $FixCapacitanceViolations()$; (Section 3.3)
3. Run STA to initialize slack and delay values for the netlist N;
4. **while** $(!S.feasible()\&\&S.leakage < best\_seen\_leakage)$ **do**
5. Update $NPaths_i$ for all cell instances c_i in the netlist N;
6. $M \leftarrow \emptyset$; $counter \leftarrow 0$;
7. **for all** cell instances, c_i in the netlist N **do**
8. **if** cell c_i is upsizable **then**
9. $m^k.target \leftarrow c_i$; $m^k.change \leftarrow upsize$;
10. $m^k.sensitivity \leftarrow \Delta TNS / \Delta leakage\_power^{\alpha}(c_i, upsize)$;
11. $M \leftarrow M \cup \{m^k\}$;
12. **end if**
13. **if** cell c_i is not a LVT cell **then**
14. $m^k.target \leftarrow c_i$; $m^k.change \leftarrow V_t$-$downscaling$;
15. $m^k.sensitivity \leftarrow \Delta TNS / \Delta leakage\_power^{\alpha}(c_i, downscale)$;
16. $M \leftarrow M \cup \{m^k\}$;
17. **end if**
18. **end for**
19. **while** $(counter < \gamma * M.size())$ **do**
20. Pick a modification m^k with maximum $sensitivity$ in M;
21. Commit $m^k.change$;
22. $M \leftarrow M \setminus \{m^k\}$;
23. $counter++$;
24. $FixCapacitanceViolations()$;
25. **end while**
26. Run STA to evaluate the current sizing solution S;
27. **end while**
28. **if** $S.feasible()$ **then**
29. Update $best\_seen\_leakage$;
30. **end if**

are sufficiently fast and robust to start with minimum-leakage configurations. Empirically, finding a timing-feasible solution in the "coarse-search" stage of our optimization accounts only for a single-percent fraction of the total required runtime (Section 4.1).

Starting with an underpowered configuration, we seek to generate feasible solutions by monotonically (i) increasing cell sizes or (ii) lowering cell threshold voltages (V_t). Both cell upsizing and V_t downscaling are performed in smallest possible steps; the ordering of these actions is determined by their *sensitivities*, which are calculated by impacts on TNS and leakage power.

$$sensitivity_{GTR} = \frac{\Delta TNS}{\Delta leakage\_power^{power\_exponent}} \quad (1)$$

When estimating the impact of a single cell modification, invoking STA can be computationally prohibitive. Therefore, we approximate the impact on TNS of a single cell modification (m_i^k) on cell c_i using $NPaths_i$, the number of negative-slack paths that pass through c_i. We define the *nearest-neighbor set* of c_i to be the set of cells that have a driver (fanin) in common with c_i, and N_i to be the union of c_i's nearest-neighbor set and c_i itself.

$$\Delta TNS(m_i^k) \approx \Sigma_{c_j \in N_i} -\Delta delay_j^k \cdot \sqrt{NPaths_j} \quad (2)$$

Here, $\Delta delay_j^k$ is an estimated delay change on c_j due to m_i^k. This approximation is based on the fact that any perturbation to cell c_i will change the delay of c_i, but also can impact the slacks of other cells that share path(s) with c_i, e.g., changing the V_t of c_i can change *required arrival times (RATs)* for its upstream cells, and change *actual arrival times (AATs)* for its downstream cells. To account for this, we introduce the factor $\sqrt{NPaths_j}$, which reflects the number of fanin and fanout cells of c_j that are affected by the delay change of c_j. If this effect is not accounted for, particularly for cells that are on critical paths, the impact on TNS will be underestimated. To

more accurately estimate the impact on TNS when we resize[1] c_i, we must consider all cells in N_i, as changing the size will affect the capacitive load on the common driver, which affects their arrival (and transition) times. Following the empirical observation in [22], we assume that the propagation of increased transition time can be safely bounded to only the nearest neighbors.

We sort the changes by non-increasing sensitivity values and commit them in order (Algorithm 1). Given that each single-cell modification is evaluated assuming other cells are fixed, the inaccuracies of sensitivity accumulate as multiple cells are changed. Therefore, we only commit the first $\gamma\%$ of the modifications between two consecutive STA invocations. The variables *power_exponent* $0 \leq \alpha \leq 3.0$ and *commit_ratio* $0 < \gamma \leq 60\%$ determine specific multistart configurations. To effectively reduce the size of the search space, we perform multilevel search (Figure 2). Fine-grain search is performed on non-dominated configurations from coarser search, but with finer steps and over smaller ranges.

3.2 Power Reduction with Feasible Timing

From the Global Timing Recovery (*GTR*) stage, we obtain a feasible sizing solution with no timing, slew or max-capacitance violations. However, the solution can improve further since some cells are oversized during the timing recovery stage. We reduce leakage power while maintaining timing feasibility by alternating (i) *sensitivity-guided greedy sizing (SGGS)*, (ii) *slack legalization*, and (iii) speeding up *bottleneck* cells.

Sensitivity-guided greedy sizing (SGGS). SGGS downsizes cells according to sensitivity while avoiding timing violations. Algorithm 2 presents pseudocode of our SGGS procedure. In this algorithm, $ISTA(c_i)$ is an incremental STA operation after cell c_i is changed. In $ISTA(c_i)$, we start timing and slack updates at the fanin nodes of the changed cell. From the fanin nodes, transition times and AATs are propagated in the forward direction, and RATs are propagated in the backward direction.

The SGGS algorithm starts with STA and initializes all timing nodes. Sensitivities are computed for all downsizable cells in Lines 3–14. We consider both gate downsizing and V_t upscaling for the sensitivity calculation. We define five sensitivities (Table 1).

Acronyms	Sensitivity functions
$SF1$	$-\Delta leakage\_power/\Delta delay$
$SF2$	$-\Delta leakage\_power \times slack$
$SF3$	$-\Delta leakage\_power/(\Delta delay \times \#paths)$
$SF4$	$-\Delta leakage\_power \times slack/\#paths$
$SF5$	$-\Delta leakage\_power \times slack/(\Delta delay \times \#paths)$

Table 1: Sensitivity functions for *SGGS*. $SF4$ and $SF5$ appear most successful (Table 4), and our metaheuristic produces better results when using multiple functions. The *slack* and $\Delta delay$ values are expected to be positive.

$\Delta leakage\_power$ and $\Delta delay$ represent leakage power and cell delay changes after the downsizing of cell c_i. The variable *slack* represents the slack at the output pin, and #*paths* is the number of timing paths that pass through the cell c_i. The *slack* value is positive since the downsizing is applied to cells with positive slack. #*paths* is calculated similarly to *NPaths* in Section 3.1, but including positive-slack paths. $SF1$ and $SF2$ have been used in [8] [12] and [11], respectively. We have added #*paths* into $SF3$ and $SF4$ to favor cells with smaller impact on the slacks of other cells. $SF5$ is a hybrid of $SF1$ and $SF2$; similar logic is used in [29], but without considering #*paths*. In Lines 15–22, we select a cell c_i with maximum sensitivity, and downsize c_i or upscale its V_t. We perform incremental timing analysis and check for violations. If the sizing step creates a timing, slew or max-capacitance violation, it is undone. The loop continues until M becomes empty.

[1] V_t-scaling does not affect the delays of neighboring cells.

Algorithm 2 Sensitivity-Guided Greedy Sizing (SGGS).

```
Procedure SGGS(N, S)
Input : sensitivity function SF, a feasible sizing solution S
Output : sizing solution S with reduced power
1.  Run STA to initialize delay values for the given solution S;
2.  M ← ∅
3.  for all cell instances, c_i in the netlist N do
4.      if cell c_i is downsizable then
5.          m^k.target ← c_i; m^k.change ← downsize;
6.          m^k.sensitivity ← ComputeSensitivity(c_i, downsize);
7.          M ← M ∪ {m^k};
8.      end if
9.      if cell c_i is not a HVT cell then
10.         m^k.target ← c_i; m^k.change ← V_t-upscaling;
11.         m^k.sensitivity ← ComputeSensitivity(c_i, upscale);
12.         M ← M ∪ {m^k};
13.     end if
14. end for
15. while M ≠ ∅ do
16.     Pick a modification m^k with maximum sensitivity in M;
17.     S' ← SaveState(S);
18.     Commit m^k.change;
19.     M ← M \ {m^k};
20.     ISTA(m^k.target);
21.     if !S.feasible() then
22.         S ← RestoreState(S');
23.     else
24.         if cell m^k.target is downsizable / not a HVT cell then
25.             Recalculate m^k.sensitivity;
26.             M ← M ∪ {m^k};
27.         end if
28.     end if
29. end while
```

Slack legalization. *ISTA* achieves a significant speedup over full-netlist STA through propagation of timing related to updated instances. We achieve further speedup by blocking the propagation when changes to timing are below a *propagation_threshold*.[2] Due to this limited accuracy, SGGS can overlook a small number of timing violations. Instead of using *GTR*, we use a *slack legalization* procedure to rectify small timing violations at a small leakage power cost.

In slack legalization, we first collect cells which are in critical (negative-slack) paths. These cells are sorted in decreasing order of $|\Delta delay|$ (delay improvement due to upsizing and V_t downscaling) and are modified in this order. Unlike *GTR*, slack legalization tracks slack changes after each cell modification, and ensures no timing degradation. Let $\Delta slack(c)$ be the slack change on output pin of cell c, and $C_{fi}(c)$ be set of fanin cells of cell c. After the modification of cell c_i, slack legalization restores the change if (i) $\Delta slack(c_i) \leq 0$, or (ii) $\Delta slack(c_i) + \Sigma_{c_j \in C_{fi}(c_i)} \Delta slack(c_j) \leq 0$. Slack legalization repeats the sizing until all timing violations are fixed.

Speeding up bottleneck cells. During greedy sizing, we size gates monotonically downward with lower-size or higher-V_t library cells, but the resulting solution can be a local optimum. We observe that a key obstacle is that we have no timing slack available to allow further gate downsizing. Therefore, to recover timing slack with the least impact to power, we speed up *bottleneck* cells, i.e., cells that participate in many timing-critical paths. We identify

[2] By default, *propagation_threshold* is set to $0.1ps$. Table 2 shows that ISTA runs faster if a higher *propagation_threshold* is given.

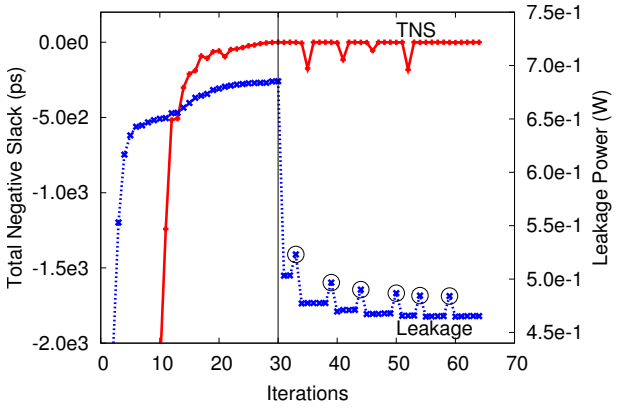

Figure 3: Progression of Power Reduction with Feasible Timing (*PRFT*) on VGA_LCD_FAST. The start of *PRFT* is marked with a vertical line; each bottleneck upsizing is marked with a circle. Slack legalization may temporarily worsen timing violations (indicated by spikes).

these cells by a bottleneck analysis similar to that provided in EDA tools [33]. We then perturb the converged solution by assigning larger sizes or lower V_t cells, and then repeat the downsizing procedure.[3] To identify bottleneck cells, we estimate total slack changes by $\Delta delay \cdot \sqrt{\#paths}$ due to hypothetical cell upsizing (or V_t downscaling). We commit the first γ% of such modifications with largest $\Delta total\_slack$, then optimize leakage power with *SGGS*. Timing violations created by speeding up bottleneck cells or *SGGS* are removed by slack legalization. To define specific multistart configurations, we sweep γ from 1% to 5% with a step size of 1%. The iterations (speeding up bottleneck cells + *SGGS* + slack legalization) terminate when the solutions stop improving. Figure 3 illustrates the progression of *PRFT*. Starting with a feasible solution from *GTR*, *PRFT* iteratively reduces leakage power while maintaining timing feasibility.

3.3 Handling Capacitance and Slew Violations

Each standard cell can drive a certain maximum capacitance load defined in the library (e.g., based on the contest library, the maximum capacitance that can be driven by the smallest four-input NAND gate with high V_t is $3.2fF$). If a cell is overloaded, its output transition time slows down significantly, resulting in overall degradation of slacks in its fanout cone. In our heuristic, we remove max-capacitance and slew violations at every iteration of *GTR* (Algorithm 1) by alternating *backward-* and *forward-traversal* repair as necessary. During *backward traversal*, we visit cells in a reverse topological order and continue to upsize driving cells until capacitance violations for the driving cells are removed or the driving cells reach their maximum sizes (whichever comes first). This procedure resolves most of capacitance violations in the early iterations of *GTR* when cell sizes are relatively small. However, at later stages, cells on certain paths can be saturated at their maximum sizes,[4] and we must downsize some of their fanout cells. Therefore, during *forward traversal*, we visit cells in a forward topological order and continue to downsize fanout cells until capacitance violations for the current cells are removed or all fanout cells shrink to their minimum sizes (whichever comes first). Empirically, this requires one to two iterations of backward and forward traversals. As this happens, all output transitions become faster than the maximum slew allowed at the ISPD 2012 contest ($300ps$).

[3]This recalls the Large-Step Markov Chain approach [21].

[4]For instance, if one inverter is driving numerous large cells, even a maximum-sized inverter cannot remove capacitance violations.

BENCHMARKS	FSTA (*sec*)	ISTA (*msec*)			
		$0ps$	$0.1ps$	$0.5ps$	$1.0ps$
DMA	0.233	1.495	0.845	0.620	0.508
PCI_BRIDGE32	0.271	0.982	0.717	0.388	0.348
DES_PERF	1.108	0.700	0.508	0.471	0.422
VGA_LCD	1.729	22.75	8.108	7.822	2.069
B19	2.435	5.460	2.717	2.115	1.833
LEON3MP	6.746	43.21	2.152	1.542	0.939
NETCARD	9.751	9.612	2.299	1.940	1.675
Geomean	924.58×	2.86×	1.00×	0.77×	0.54×

Table 2: Runtime comparisons of full-scale STA (sec) and incremental STA (msec) after changing the size of one cell (averaged over 100,000 randomly selected independent experiments). *ISTA* is tested by varying *propagation_threshold*.

4. EMPIRICAL VALIDATION

Our implementation Trident is written in C++, compiled with g++ 4.6.2 and validated on a 3.2GHz Intel Xeon E31230 Linux workstation with 8GB of memory, using four CPU cores. We compare it to the results of the ISPD 2012 Gate Sizing Contest on the ISPD 2012 benchmark suite [23]. Timing violations are verified by PrimeTime, and leakage-power values are read from the official contest evaluation script [23]. In all experiments, we use the default setting described in Section 3.

4.1 Analysis of Our Implementation

Trident is a stand-alone tool that includes a built-in static timer and relies only on standard C++ libraries. Instead of analytical model-fitting, the built-in timer is based on library table lookups, linear interpolation, and timing propagation. With capacitive modeling of wires used in the contest, the timer correlates to PrimeTime within $10^{-3}ps$ precision but runs 60× faster on average. Table 2 compares the runtimes of *ISTA* and full-scale STA of our timer.

Running in 3-4 threads, Trident generates feasible solutions for all 14 benchmarks in 83 hours using less than 6.0GB of memory. Our implementation dynamically assigns multistart configurations to available threads, and therefore it can readily issue more parallel threads as memory allows. An example runtime breakdown on the NETCARD_SLOW benchmark is as follows. *GTR* takes 31.3% (6.2% coarse search, 25.1% fine-grain search) of total runtime, of which 53% is spent in full-scale STA, 20% in TNS estimation, and 12% is spent in *FixCapViolation*. *PRFT* takes 68.5% (45.4% SGGS, 23.0% perturbing iterations), of which 87.6% is in *ISTA*. I/O takes 0.2% of runtime.

4.2 Comparisons to the State of the Art

Table 3 compares Trident to top contestants at the ISPD 2012 Gate Sizing Contest. Performance results[5] for individual teams are quoted from [23]. Trident has found feasible sizing solutions for all circuits in the ISPD 2012 benchmark suite. Compared to the top three teams, Trident achieves the lowest leakage power for 13 out of 14 circuits (no parameter tuning to specific benchmarks has been employed). On average, Trident obtains leakage power improvement of 43%, 16%, 52% versus NTUgs (National Taiwan University), UFRGS-Brazil (Universidade Federal do Rio Grande do Sul), and PowerValve (National Tsing Hua University and Missouri University of S&T), respectively. Geometric means are calculated excluding infeasible benchmarks, which underrepresents the impact of our proposed techniques. All of our runs are finished within the corresponding hard runtime limits [23]. For further analysis, we provide the best parameter values found by our metaheuristics for individual benchmarks in Table 4.

[5]The contest considered only leakage and not dynamic power.

BENCHMARKS	(# OF CELLS)	ISPD 2012 contest results (leakage power)				Trident		
		NTUgs	UFRGS-Brazil	PowerValve	Best	Leakage power after *GTR*	Leakage power after *PRFT*	Wallclock time
DMA_FAST	(25.3K)	0.511	0.323	0.312	0.312	0.650	**0.299**	13.9
DMA_SLOW	(25.3K)	0.205	0.158	0.147	0.147	0.211	**0.145**	9.9
PCI_BRIDGE32_FAST	(33.2K)	0.512	**0.168**	0.226	**0.168**	0.348	0.183	13.0
PCI_BRIDGE32_SLOW	(33.2K)	0.203	0.115	0.116	0.115	0.185	**0.111**	10.2
DES_PERF_FAST	(111K)	2.390	3.520	2.320	2.320	7.157	**1.842**	82.7
DES_PERF_SLOW	(111K)	0.674	0.884	0.697	0.674	0.922	**0.614**	70.1
VGA_LCD_FAST	(165K)	0.758	0.580	0.773	0.580	0.685	**0.471**	45.6
VGA_LCD_SLOW	(165K)	0.415	0.378	0.391	0.378	0.454	**0.351**	87.5
B19_FAST	(219K)	2.710	–	4.490	1.040	1.377	**0.771**	206.5
B19_SLOW	(219K)	0.627	0.614	0.736	0.614	0.718	**0.583**	213.9
LEON3MP_FAST	(649K)	–	–	4.940	2.020	1.989	**1.487**	1323.2 *
LEON3MP_SLOW	(649K)	1.420	1.790	2.960	1.420	1.422	**1.341**	1274.0
NETCARD_FAST	(959K)	2.010	2.300	2.970	2.010	1.997	**1.861**	1096.9
NETCARD_SLOW	(959K)	1.770	1.970	1.940	1.770	1.818	**1.770**	299.9
GEOMETRIC MEAN		**1.43**×	**1.16**×	**1.52**×	**1.11**×	**1.53**×	**1.00**×	

Table 3: Leakage power (W) and wall clock time (in minutes) on ISPD 2012 benchmarks. Experiments have been performed on a 3.2GHz Intel Xeon E31230 Linux workstation with 8GB of memory, using four CPU cores (and four threads). The sizing solutions are verified by PrimeTime and the official contest evaluation script [23]. Geometric means are calculated excluding infeasible solutions, which are marked with "–". Results of LEON3MP_FAST(∗) are reported before completing the perturbing (upsizing) iterations in *PRFT* due to runtime limitations imposed by the ISPD 2012 experimental protocol.

Comparing our approach to [22], we note the following.

- The winners of the contest (NTUgs) have implemented algorithms in [22] with additional improvements, and we include their contest results in Table 3.
- Intel released their results for five benchmarks (out of 14 total), where they observed significant room for improvement compared to the best contest results. Of these five benchmarks, we outperform [23] on four, and are slightly behind on one.

Hence, our optimizer appears competitive.

4.3 Comparison to Minimum-Leakage Solutions

In addition to reporting achievable results, we estimate available room for further improvement. Starting with minimum-leakage configurations for each cell instance, we heuristically fix slew and max-capacitance violations while increasing leakage power by 5.8% (NETCARD) to 53.5% (DMA). In other words, we estimate the leakage cost of achieving electrical feasibility with respect to only load and slew constraints (ignoring timing). The ratio of total leakage power of our timing-feasible configurations to that in solutions constructed as just described gives an indication of the additional leakage penalty needed to fix timing violations. Table 5 shows that for the largest benchmarks the penalty is very small, and our solutions likely cannot be improved by more than several percent. Benchmarks with tighter timing constraints ("fast") require greater leakage penalty to achieve timing feasibility, and this is especially true for smaller benchmarks. This suggests that the availability of a strong gate-sizer could allow the tightening of timing constraints for larger circuits. Based on leakage power in Table 3, the last column in Table 5 approximates the amount of total negative slack that can be removed by each doubling of leakage power from the electrically feasible solutions that we have constructed.

BENCHMARKS	*GTR*		*PRFT*	
	α	γ(%)	*SF#*	γ(%)
DMA_FAST	0.91	24.5	SF5	1.0
DMA_SLOW	1.00	10.0	SF5	5.0
PCI_BRIDGE32_FAST	0.91	34.0	SF4	4.0
PCI_BRIDGE32_SLOW	1.11	36.0	SF5	4.0
DES_PERF_FAST	0.85	46.5	SF5	1.0
DES_PERF_SLOW	0.83	8.5	SF2	3.0
VGA_LCD_FAST	0.70	17.5	SF5	4.0
VGA_LCD_SLOW	1.00	10.0	SF4	3.0
B19_FAST	1.33	16.5	SF2	4.0
B19_SLOW	1.50	7.5	SF5	1.0
LEON3MP_FAST	0.71	7.0	SF4	1.0
LEON3MP_SLOW	0.89	4.0	SF4	2.0
NETCARD_FAST	0.57	4.0	SF3	1.0
NETCARD_SLOW	2.67	4.0	SF3	1.0

Table 4: Parameters for *GTR* and *PRFT* associated with the best solutions found. This parameter sweep is included in reported runtimes. *SF*1-*SF*5 are described in Table 1.

5. CONCLUSIONS

Thirty years of research on gate sizing have generated a large number of interesting ideas, but leave unclear how to write a competitive gate-sizer. Significant improvements made by recent industry tools [22] suggest that newer, more powerful optimization methods could find even better power-performance tradeoffs in practice. This possibility has been confirmed at the ISPD 2012 Gate Sizing Contest, organized by researchers from Intel [23], where none of the contestants dominated on the entire benchmark set. Each team excelled on a small subset of benchmarks, and the best results

BENCHMARKS	Min. leak.	Ratio	$\frac{\Delta TNS}{log_2 Ratio}$ (μs)
DMA_FAST	0.073	4.08	1.92
PCI_BRIDGE32_FAST	0.063	2.89	1.95
DES_PERF_FAST	0.268	6.88	1.31
VGA_LCD_FAST	0.303	1.54	31.2
B19_FAST	0.522	1.48	11.0
LEON3MP_FAST	1.311	1.15	148
NETCARD_FAST	1.766	1.06	216
DMA_SLOW	0.073	1.98	3.40
PCI_BRIDGE32_SLOW	0.063	1.75	3.25
DES_PERF_SLOW	0.268	2.29	2.58
VGA_LCD_SLOW	0.303	1.16	77.4
B19_SLOW	0.522	1.12	24.2
LEON3MP_SLOW	1.311	1.02	643
NETCARD_SLOW	1.766	1.002	2290

Table 5: Leakage power ratios of electrically feasible solutions to our best solutions. Since our solutions are (timing) feasible, ΔTNS equals TNS of electrically feasible solutions.

on some benchmarks have been produced by teams not in the top three. These data reflect the importance and complexities of the (discrete) gate-sizing problem, as well as the amount of room for further improvement, despite significant recent progress.

The most sophisticated published techniques for gate sizing are analytical in nature and assume continuous sizing and/or V_t assignment, and sometimes implicitly assume convexity in their optimization approach. These techniques can be frustrated by the combinatorial nature of discrete sizing and by the nonconvexity of circuit delay caused by side capacitance and tabular delay lookups. Combinatorial techniques can also be found in the literature, but either do not scale to large circuits (e.g., branch-and-bound) or remain limited to greedy optimization, which can become stuck in local optima due to the nonconvexity of delay. Significant progress has been recently achieved using dynamic programming, e.g., by some ISPD 2012 contestants [22]. However, these techniques may be less efficient on circuits with significant reconvergence, and may require long runtimes for large circuits or libraries.

In this work, we observe that gate sizing retains some aspects of convexity *in the global sense*, and that carefully prioritized greedy optimization can bring significant improvement. To this end, we develop several insights into (*i*) the sensitivity functions that lead to effective prioritization of gate upsizing and (*ii*) how these functions can be implemented efficiently. The best configurations of sensitivity functions apparently depend on circuit structure (depth, width, number of paths, etc.). Therefore, we parameterize the space of sensitivity functions, and develop metaheuristics that traverse this space by independently invoking lower-level heuristics at individual points. This optimization leverages *sequential importance sampling* from statistical physics [9], in the form of the *go-with-the-winners (GWTW)* metaheuristic that was previously analyzed in [1] and shown to perform on par with simulated annealing. To make this approach practical, we develop high-performance implementations of individual heuristics.

Empirical results on ISPD 2012 benchmarks, following the ISPD 2012 contest protocol, show that our implementation outperforms the best results recorded at the contest for all but one benchmark. Our implementation outperforms each individual contestant by a large margin, but by no means does it give the last word on the subject. Rather, many opportunities opened by our research remain unexplored, and we foresee that empirical performance can be improved further. Such improvements, as well as the ones reported in our work, will significantly enhance the power-performance tradeoffs in future generations of integrated circuits.

6. REFERENCES

[1] D. Aldous and U. Vazirani, "'Go with the Winners' Algorithms", *Proc. FOCS*, 1994, pp. 492–501.

[2] 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.

[3] K. D. Boese, A. B. Kahng and S. Muddu, "A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations", *Operations Research Letters* 16(2) (1994), pp. 101–113.

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

[5] D. G. Chinnery and K. Keutzer, "Linear Programming for Sizing, V_{th} and V_{dd} Assignment", *Proc. ISLPED*, 2005, pp. 149–154.

[6] H. Chou, Y.-H. Wang, and C. C.-P. Chen, "Fast and Effective Gate Sizing with Multiple-V_t Assignment Using Generalized Lagrangian Relaxation", *Proc. ASP-DAC*, 2005, pp. 381–386.

[7] O. Coudert, "Gate Sizing for Constrained Delay/Power/Area Optimization", *IEEE Trans. on VLSI Systems* 5(4) (1997), pp. 465–472.

[8] J. P. Fishburn and A. E. Dunlop, "Tilos: A Posynomial Programming Approach to Transistor Sizing", *Proc. ICCAD*, 1985, pp. 326–328.

[9] T. F. Gonzalez (editor), *Handbook of Approximation Algorithms and Metaheuristics*, Chapman and Hall/CRC 2007.

[10] P. Grassberger, "Go with the Winners: A General Monte Carlo Strategy", `http://arxiv.org/pdf/cond-mat/0201313v1.pdf`.

[11] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Selective Gate-Length Biasing for Cost-Effective Runtime Leakage Control", *Proc. DAC*, 2004, pp. 327–330.

[12] P. Gupta, A. B. Kahng and P. Sharma, "A Practical Transistor-Level Dual Threshold Voltage Assignment Methodology", *Proc. ISQED*, 2005, pp. 421–426.

[13] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Gate-Length Biasing for Runtime-Leakage Control", *IEEE Trans. on CAD* 25(8) (2006), pp. 1475–1485.

[14] S. Held, "Gate Sizing for Large Cell-Based Designs", *Proc. DATE*, 2009, pp. 827–832.

[15] S. Hu, M. Ketkar and J. Hu, "Gate Sizing for Cell-Library-Based Designs", *IEEE Trans. on CAD* 28(6) (2009), pp. 818–825.

[16] IWLS 2005 Benchmarks, `http://iwls.org/iwls2005/benchmarks.html`

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

[18] W. N. Li, "Strongly NP-Hard Discrete Gate-Sizing Problems", *IEEE Trans. on CAD* 13(8) (1994), pp. 1045–1051.

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

[20] N. D. MacDonald, "Timing Closure in Deep Submicron Designs", *DAC Knowledge Center Article*, 2010.

[21] O. Martin, S. W. Otto and E. W. Felten, "Large-Step Markov Chains for the Traveling Salesman Problem", *Complex Systems* 5(3) (1991), pp. 299–326.

[22] M. M. Ozdal, S. Burns and J. Hu, "Gate Sizing and Device Technology Selection Algorithms for High-Performance Industrial Designs", *Proc. ICCAD*, 2011, pp. 724–731.

[23] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and C. Zhuo, "ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ISPD*, 2012, pp. 161–164. http://archive.sigda.org/ispd/contests/12/ispd2012_contest.html

[24] M. Rahman, H. Tennakoon and C. Sechen, "Power Reduction via Near-Optimal Library-Based Cell-Size Selection", *Proc. DATE*, 2011, pp. 867–870.

[25] H. Ren and S. Dutt, "A Network-Flow Based Cell Sizing Algorithm", *Proc. IWLS*, 2008, pp. 7–14.

[26] S. Roy, W. Chen, C. C.-P. Chen and Y. H. Hu, "Numerically Convex Forms and Their Application in Gate Sizing", *IEEE Trans. on CAD* 26(9) (2007), pp. 1637–1647.

[27] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya and S.-M. Kang, "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization", *IEEE Trans. on CAD* 12(11) (1993), pp. 1621–1634.

[28] S. Shah, A. Srivastava, D. Sharma, D. Sylvester, D. Blaauw and V. Zolotov, "Discrete V_t Assignment and Gate Sizing Using a Self-Snapping Continuous Formulation", *Proc. ICCAD*, 2005, pp. 704–711.

[29] A. Srivastava, D. Sylvester and D. Blaauw, "Power Minimization Using Simultaneous Gate Sizing, Dual-V_{dd} and Dual-V_{th} Assignment", *Proc. DAC*, 2004, pp. 783–787.

[30] H. Tennakoon and C. Sechen, "Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-Based Pre-Processing Step", *Proc. ICCAD*, 2002, pp. 395–402.

[31] J. Wang, D. Das and H. Zhou, "Gate Sizing by Lagrangian Relaxation Revisited", *IEEE Trans. on CAD* 28(7) (2009), pp. 1071–1084.

[32] L. Wei, K. Roy and C. Koh, "Power Minimization by Simultaneous Dual-V_{th} Assignment and Gate-Sizing", *Proc. CICC*, 2000, pp. 413–416.

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