

A Unified Discrete Gate Sizing/Cell Library Optimization Method for Design and Analysis of Delay Minimized CMOS and BiCMOS Circuits

Kerry S. Lowe and P. Glenn Gulak

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 1A4

Abstract

This paper presents the first reported discrete gate sizing method to jointly include library optimization capability. The method enables a designer to find the best set of sizes to include in a library and study the trade-off between the number of gate sizes in a library and circuit performance. Compared with continuous sizing, discrete sizing with library optimization, achieves within 2% speed performance using 2X to 5X fewer cells in 8-bit adders.

1 Introduction

Gate sizing optimization strives to improve the (post-layout) delay performance of a given combinational logic circuit through the judicious adjustment of gate sizes consistent with the set of permitted sizes and other possible constraints such as on maximum circuit power. In the case that permitted sizes span a continuous range, the sizing problem is efficiently solved and the resulting performance improvement is optimal [1-2]. However, the resulting optimized design may be impractical or expensive to implement since a large number of different sizes may be needed each necessitating a customized gate layout. Consequently, a common practice is to *discretely* optimize the circuit so that sizes in the optimized design are restricted to those that are available in a predefined library (i.e. semi-custom design). In this case, careful consideration is needed as to the composition of the library (i.e. how many and what sizes to include) as well as to the size selection for each gate in the circuit since both of these factors strongly effect circuit performance.

To date, research on discrete optimization has concentrated on the latter consideration treating the library as a given [3-5]. No systematic effort appears to have been directed at studying the trade-off between the number of allowed gate sizes in a library and optimized circuit performance or at determining the best set of size values to include in a library. Previous research has also not directly addressed the problem of discrete optimization with a library that contains both gates and

buffers. This capability is useful when optimizing a BiCMOS technology logic circuit since a BiCMOS gate can typically be modeled as a buffered CMOS gate [6].

This paper aims to address these limitations and provide a comprehensive capability for the sizing optimization of logic circuits by formulating and solving a generalized sizing problem that incorporates the above cases. The problem is labelled the joint *gate sizing/library optimization* problem and concerns finding, for a given logic network, the set of sizes and optimized network design such that network delay is minimized subject to a library cost constraint (in terms of the number of unique cells used) and power constraint. The proposed solution method combines precise continuous sizing techniques along with heuristics and unifies the treatment of discrete and continuous optimization (i.e. since optimization results converge to the continuous case result if the number of allowed sizes corresponds to the continuous case requirement). As well, the method enables the efficient determination of the trade-off between the number of allowed sizes and performance (i.e. since only one continuous optimization is needed in determining this trade-off curve) and is applicable to the cases of buffer insertion optimization and BiCMOS design.

2 The Sizing/Library Optimization Problem

In a *gate sizing/library optimization* problem, the logic circuit to be optimized is represented as a directed acyclic graph Γ (e.g. Fig. 1).

Definition 1: A combinational logic network is a directed acyclic graph $\Gamma = (\{v\}, \{e\}, \{C_w\})$. The set of vertices is given by $\{v\} = \{k\} \cup \{n\} \cup \{m\}$ where $\{k\}$ is the set of K gates, $\{n\}$ is the set of N network inputs, and $\{m\}$ is the set of M network outputs. Each vertex $k \in \{k\}$ implements a specified single output boolean logic primitive. There are \wp such primitives in Γ . There is an edge $(k, j) \in \{e\}$ if and only if the output of vertex k is connected with an input of vertex j . For each vertex $k \in$

$\{k\}$ there is an associated $c_w^k \in \{c_w\}$. The term c_w^k is the net wire capacitance of the edges $(k, j \in \{j\}^k)$ where $\{j\}^k$ is the set of vertices connected to the output of k .

Different realizations of logic gate k in Γ correspond to a different choice of gate size and/or logic family. (e.g. CMOS logic family or BiCMOS logic family). The size of gate k is denoted by s^k while the logic family of gate k is denoted ψ^k . A set of selections for the size and logic family for each of the K gates in Γ is written as $\{\psi^k\}, \{s^k\}$ and thus defines a specific implementation of network Γ . Note that a buffered (or BiCMOS) gate is typically modeled as a 2-stage gate (Fig. 2).

Definition 2: The size of gate k , s^k , specifies its transistor sizes relative to the minimum allowed value. The dimension of s^k is dependent on the logic family of gate k and is equal to the number of independent stages that are deemed to comprise gate k . Independent stage x is a subcircuit of gate k whose transistors may be uniformly scaled by a relative factor s_x^k such that all node voltage levels are preserved. If gate k has X stages then $s^k = (s_1^k, s_2^k, \dots, s_X^k)$ where the stages are numbered in increasing order from the input (stage 1) to the output (stage X). The stage sizes s_x^k are taken to be $1 \leq s_x^k \leq s_{max}$ where s_{max} is an upper size limit. In sizing a gate, only the FET gate widths and bipolar emitter lengths are scaled. FET lengths and emitter widths are kept constant.

The aggregate cell library Ψ is the set of all (unique) logic cells (of different size, logic primitive, or logic family) that is required for a specific implementation of network Γ . The number of cells in Ψ is denoted by \aleph and is a measure of the complexity or cost of the library that is needed to realize Γ . \aleph is calculated from the cell allocation vector $\underline{\aleph}$ whose elements list the number of unique gates sizes for each primitive in Γ .

Definition 3: The number of unique cells \aleph used in a specific implementation of Γ is defined as

$$\aleph = \sum_{y=1}^{p+1} \aleph_y \quad (1)$$

where p is the number of unique primitives in Γ , \aleph_y is the number of unique gate cells that implement primitive y , and \aleph_{p+1} is the number of unique buffer cells. The term $\underline{\aleph} = [\aleph_1, \aleph_2, \dots, \aleph_p; \aleph_{p+1}]$ is the cell allocation vector.

The delay T of network Γ represents the objective function to be minimized. In this paper, T is calculated assuming a linear loaded gate delay model that contains both fixed delay and fan-out dependent delay.

Definition 4: The delay T of network Γ is

$$T = \max_{\Phi} \left[\sum_{k \in \{k\}_{\Phi}} \tau^k \right] \quad (2)$$

where $\{k\}_{\Phi}$ is the set of gates in the Φ 'th distinct delay path ($\Phi = 1, 2, \dots, \Phi$) that starts at an input vertex and ends at an output vertex. τ^k denotes the worst case delay of gate k and is calculated as

$$\tau^k = \alpha^k + \gamma^k \xi^k = \alpha^k + \gamma^k (C_{out}^k / C_{in}^k) \quad (3)$$

where α^k is the internal delay, γ^k the external delay factor,

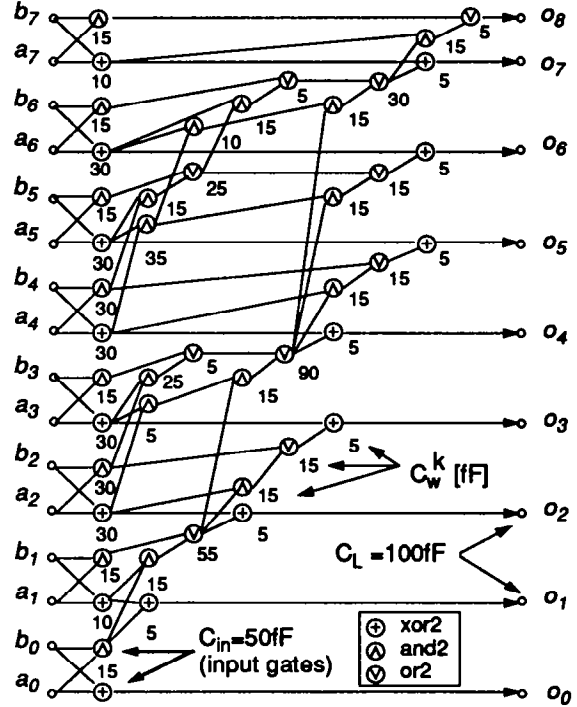


Fig. 1: Graph representation of 8-bit CLA adder network ($\Gamma = \text{add8a}$)

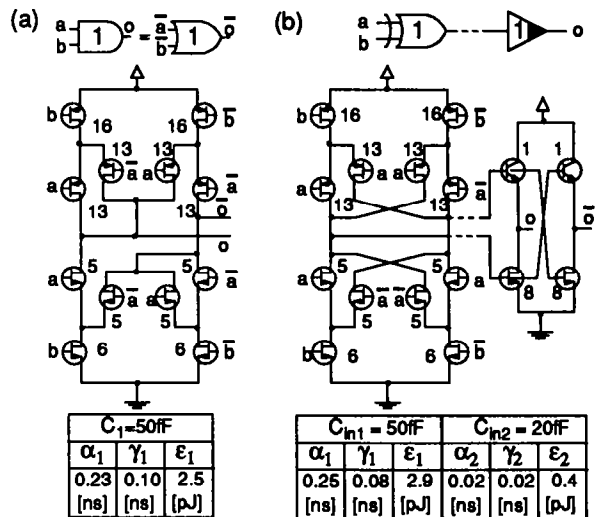


Fig. 2: Example cell designs and model, (a) 1-stage CMOS gate, (b) 2-stage BICMOS gate

and ξ^k the fan-out. For a X -stage gate, α_x^k , γ_x^k and $C_{in_x}^k$ refer to a unit size ($s_x^k = 1$) stage x of gate k and

$$\alpha^k = \sum_{x=1}^X \alpha_x^k + \sum_{x=1}^{X-1} \gamma_x^k \frac{C_{in_{x+1}}^k s_{x+1}^k}{C_{in_x}^k s_x^k}, \quad (4)$$

$$\gamma^k = \gamma_x^k \frac{C_{in_1}^k s_1^k}{C_{in_x}^k s_x^k}, \text{ and } C_{in}^k = C_{in_1}^k s_1^k. \quad (5)$$

In calculating network power P , a linear model is assumed with gate power proportional to gate size.

Definition 5: The power P of network Γ refers to either the static power P_0 or dynamic dissipation rate P_d . P_0 is

$$P_0 = \sum_{k=1}^K \sum_{x=1}^{X_k} p_{0_x}^k s_x^k \quad (6)$$

where $p_{0_x}^k$ is the $s_x^k = 1$ static power for stage x of gate k . P_d is given by (7) where ΔV_{out}^k is the output voltage swing of gate k , η^k is its transition density [12], and ϵ_x^k is the internal energy required to change the state of a unit size stage x of gate k ($s_x^k = 1$). If $x > 1$, ϵ_x^k includes the energy required to charge/discharge its input capacitance.

$$P_d = \sum_{k=1}^K \left[C_{out}^k \eta^k [\Delta V_{out}^k]^2 + \sum_{x=1}^{X_k} (\eta^k \epsilon_x^k s_x^k) \right]. \quad (7)$$

Thus, with reference to Definitions 1-5, the discrete gate sizing/library optimization problem of Fig. 1 is stated formally by Problem 1. Note that Problem 1 reduces to 1) a continuous range sizing problem if no $\aleph \leq \aleph_{max}$ constraint is imposed (or if $\aleph_{max} \rightarrow K$) and 2) a unconstrained power problem if no $P \leq P_{max}$ constraint is imposed (or if $P_{max} \geq P_{\aleph}$ where P_{\aleph} is the power at minimum delay with \aleph cells). (A fixed library discrete sizing problem is obtained by altering the size range constraints e.g. $s_x^k \in \{s_{min}^k, s_{min+1}^k, \dots, s_{max}^k\}$).

Problem 1 : Discrete gate sizing / library optimization problem

Given: Network Γ , P_{max} , s_{max}^k , s_{min}^k , and \aleph

Find: The implementation $\{\Psi^k\}, \{s_x^k\}$ that minimizes delay T

Such that: $\aleph \leq \aleph_{max}$, $P \leq P_{max}$, $s_{min}^k \leq s_x^k \leq s_{max}^k$, $\forall s_x^k$.

where \aleph_{max} is defined by \aleph using (1).

3 Problem solution method

The proposed solution method for Problem 1 decomposes Problem 1 into three optimizations or major steps (Fig. 3). First, step (a) finds the continuous case sizing solution for network Γ using the method of [2]. In this method, gate (or gate and buffer) sizes are found by formulating and solving a posynomial program (or

sequence of posynomial programs) since such optimization programs have the desirable property that each can be solved with global optimality [7-8]. Specifically, a procedure called *Posy* $[g_o, \{g_c \leq 1\} \rightarrow \{z\}]$ is utilized that finds the values for the design variables (i.e. sizes) $\{z\}$ that minimizes the posynomial objective function g_o subject to the posynomial constraints $\{g_c \leq 1\}$.

Second, step (b) imposes the discrete case \aleph_{max} constraint and assigns a size category to each gate. (There are a total of \aleph_{max} categories). This is done by using step (a) results (Fig. 3a) and considering the distribution of the relative importance or weight of each gate k versus

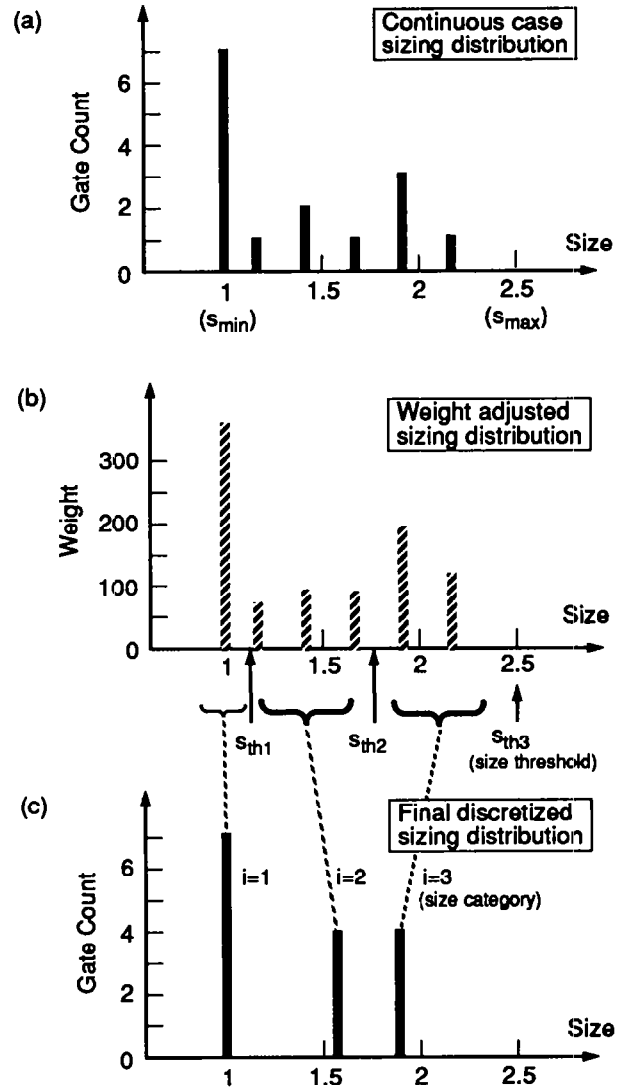


Fig. 3: An illustration of 3-step approach used to solve Problem 1: (a) gate size distribution (for a given primitive) determined from continuous case optimization, (b) weighted distribution and size categorization reducing number of sizes from 6 to 3, (c) final sizing

size (Fig. 3b). This weight is defined as

$$\omega^k = \sum_{\Phi \in \Phi^k} \left(\sum_{j \in U_j} \tau^j \right) \quad (8)$$

where $\Phi^k \subseteq \Phi$ is the set of delay paths that contain gate k . The weight of a gate, as given by (8), captures the fact that important gates (or vertices in Γ) belong to long paths or many paths, or both.

From the weight distribution, a figure of merit called the *expected deviation* $\Lambda(\underline{N})$ from the continuous case solution is defined and used to determine the most appropriate assignment of *size categories* to gates. Specifically, $\Lambda(\underline{N})$ is defined by

$$\Lambda(\underline{N}) = \sum_{y=1}^{p+1} \Lambda_y \quad (9)$$

$$\Lambda_y = \min_{\forall \{s_{th}\}_y} \sum_{i=1}^{N_y} \sum_{k \in \{i\}} \omega^k (s^k - \hat{s}_i)^2 \quad (10)$$

where $i \in \{1, 2, \dots, N_y\}$ is the *size category* for each gate k of primitive y in Γ as determined by the set of size threshold settings $\{s_{th}\}_y = \{s_{th1}, s_{th2}, \dots, s_{thN_y}\}$. The term \hat{s}_i in (10) is the mean size of the gates in category i .

The set of gates of primitive y belonging to size category i is denoted by $\{i\}$ and each such gate has a size s^k such that $s_{th(i-1)} < s^k \leq s_{thi}$ where $s_{th0} = 0$. The minimization in (10) is taken over all threshold settings that yield a unique partitioning of the N_c distinct continuous case sizes into N_y ordered categories such that relative ordering of sizes is preserved and each category contains at least one size. The minimization is done by exhaustive search since each candidate partition is simply evaluated and size rounding limits N_c and hence combinatorial complexity. If an allocation \underline{N} is not specified, step (b) can also attempt to find an optimal allocation. This is done by finding and comparing the *expected deviation* $\Lambda(\underline{N})$ for each valid allocation.

Finally, step (c) of the solution method for *Problem 1* finds the sizing solution $\{s^k\}$ (and hence size for each category i) consistent with the allocation and threshold settings of step (b). The task is accomplished by re-employing *Posyl 1* with the added constraint that gates from the same category have the same size (Fig. 3c).

The solution method for *Problem 1* is implemented in a program named TOP (Technology Optimization Program), Fig. 4. TOP includes a capability for systematically changing the \underline{N} and P_{max} settings of *Problem 1* and solves the *Posyl 1* task using the ADS routine [9]. Note that the (continuous case) delay from step (a) provides a lower bound value for assessing the optimality of the final output and, in particular, the heuristic (8)-(10) of step (b).

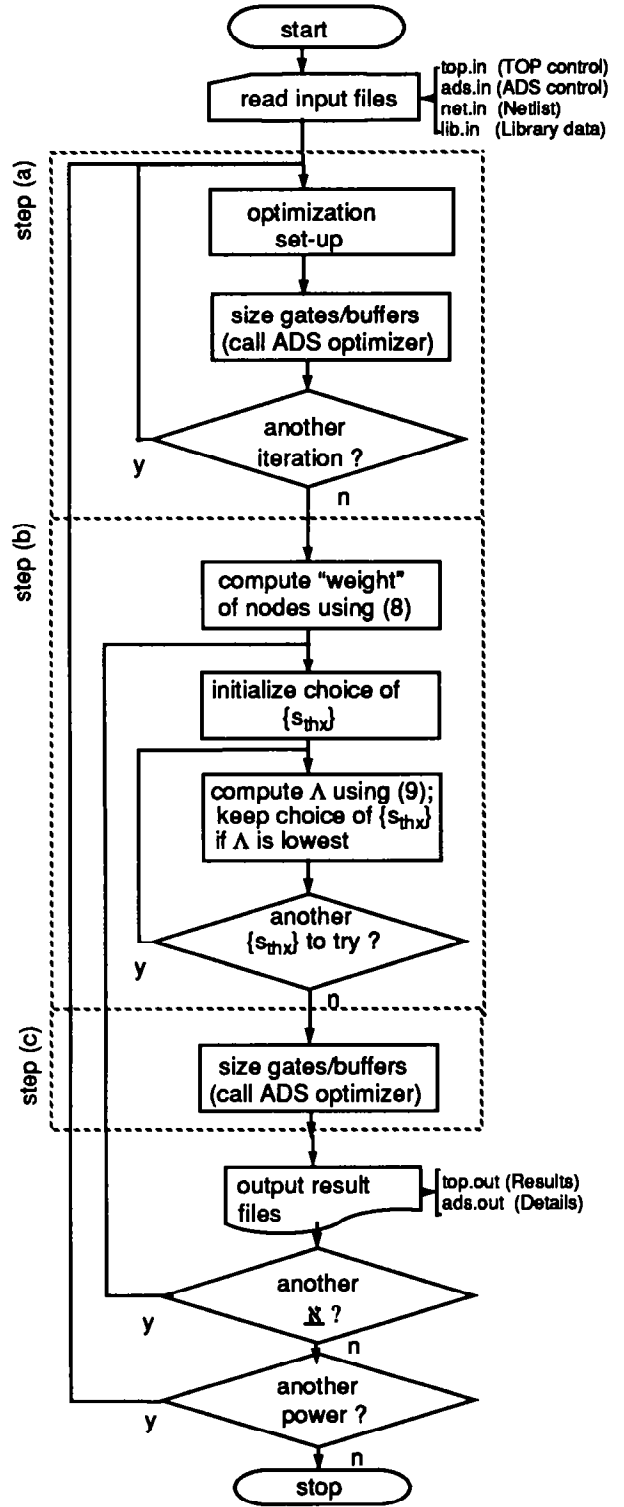


Fig. 4: Implementation of *Problem 1* solution method (the TOP program)

4 Experimental Results

Optimization results for the example 8-bit lookahead carry adder [10] of Fig. 1 are shown in Fig. 4. These results use the CMOS and BiCMOS gate data of Fig. 2 based on a 0.8 μ m process [11]. Fig. 5a shows the delay versus power performance for different choices of allocation. Note that as the number of allocated sizes is increased, performance approaches the continuous case. This behavior is also evident from Fig. 5b that plots T vs. number of allocated sizes for the case $P_{\max}=1.5P_u$. (The term P_u (T_u) is the power (delay) when all gates are minimum size.) Moreover, it is seen that discrete case delay rapidly converges to the continuous case delay. In the CMOS case just 4 sizes of gate (per primitive) are needed to achieve speed performance that is within 2% of

the continuous case. The cell count savings is 2X (assuming continuous case sizes are 0.1 rounded). In the BiCMOS case, just 4 sizes of gates and buffers are needed to come within 2% speed performance. The cell count savings is 3.4X. The optimized gate and buffer sizes (for the BiCMOS case) are summarized in Fig. 5c-f. Continuous case sizes are shown in Fig. 5c and Fig. 5e. Discrete case sizes for the allocation of 2 sizes (of gates and buffers) are given in Fig. 5d and Fig. 5f. Note that as the P_{\max} setting is varied, TOP judiciously adjusts discrete case gate sizes maintaining discrete case performance close to continuous case performance (Fig. 5a) and the potential for cell count savings.

Other results are summarized in Table 1 and further demonstrate the utility of the proposed method for the

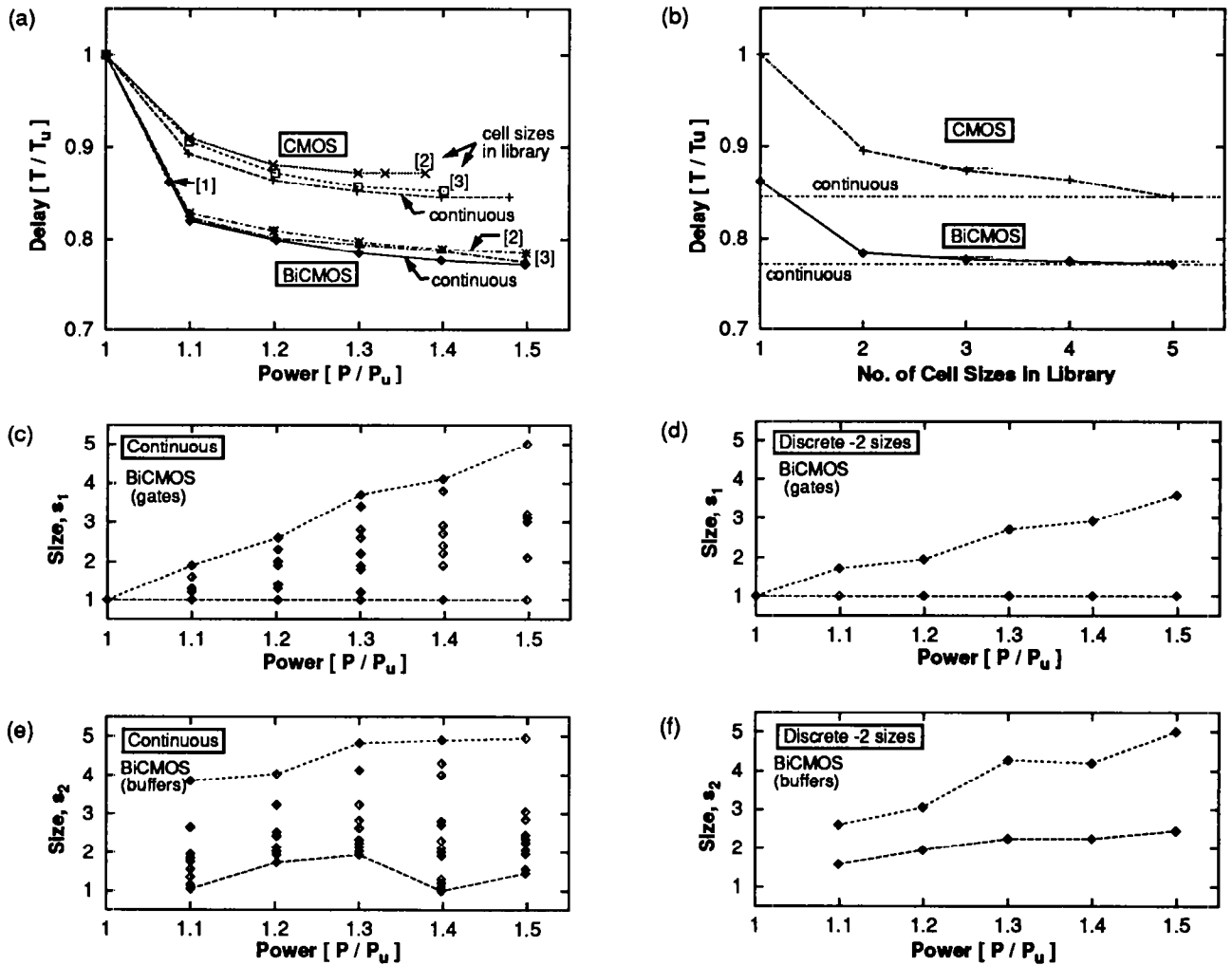


Fig. 5: Optimization results for network of Fig. 1 using CMOS and BiCMOS cell data of Fig. 2 (for $s_{\max} = 5$) (a) delay vs power, (b) delay vs number of allowed sizes per (gate or buffer) primitive, (c-f) size values for continuous sizing optimization and discrete sizing with 2 allowed sizes per primitive.

Table 1: Number of discrete case sizes needed to achieve delay within given percent of (unconstrained power) continuous case delay. (CPU times are for Sun 4/40 SPARCstation IPC with 20MB MM.)

	Network	K nodes	ϕ prim.	Cpu [s]	Continuous Case		Discrete Case - Required Allocation, \mathbb{K}		
					T [ns]	Allocation, \mathbb{K}	$\leq 5\%$	$\leq 2\%$	$\leq 1\%$
CMOS	par16a (parity)	9	1	2	2.21	[3]	[2]	[2]	[2]
	par16b (parity)	15	1	5	1.41	[4]	[2]	[2]	[2]
	dec16a (bal. decoder)	32	1	8	1.14	[3]	[2]	[2]	[2]
	dec16b (tree decoder)	36	1	14	1.46	[4]	[2]	[2]	[3]
	add8d (ripple)	37	2	26	5.62	[9,1]	[2,1]	[2,1]	[3,1]
	add8c (intermediate)	40	2	28	4.90	[12,1]	[2,1]	[2,1]	[3,1]
	add8b (intermediate)	43	2	42	4.18	[11,1]	[2,1]	[3,1]	[3,1]
	add8a (lookahead)	46	2	38	3.49	[9,1]	[2,1]	[4,1]	[5,1]
	mult8a (parallel mult.)	112	2	200	10.23	[15,16]	[3,3]	[5,5]	[9,9]
BiCMOS	par16a (parity)	9	1	5	2.06	[3; 3]	[2; 2]	[2; 2]	[2; 2]
	par16b (parity)	15	1	18	1.39	[3; 1]	[2;1]	[2;1]	[2;1]
	dec16a (bal. decoder)	32	1	48	0.99	[2; 3]	[2; 2]	[2; 2]	[2; 2]
	dec16b (tree decoder)	36	1	60	1.33	[3; 4]	[2; 2]	[2; 2]	[3; 3]
	add8d (ripple)	37	2	213	5.34	[9, 1; 10]	[2, 1; 2]	[2, 1; 2]	[4, 1; 4]
	add8c (intermediate)	40	2	180	4.63	[11, 1, 14]	[2, 1; 2]	[2, 1; 2]	[4, 1; 4]
	add8b (intermediate)	43	2	225	3.89	[10, 1; 10]	[2, 1; 2]	[2, 1; 2]	[4, 1; 4]
	add8a (lookahead)	46	2	244	3.19	[6, 1; 10]	[2, 1; 2]	[2, 1; 2]	[3, 3; 3]
	mult8a (parallel mult.)	112	2	2232	9.07	[11,12; 26]	[2, 2; 2]	[3, 3; 3]	[4, 4; 4]

design and analysis of CMOS and BiCMOS circuits. In 8-bit adder and 8x8 bit multiplier networks, for example, it is seen that discrete sizing with library optimization achieves within 2% of the continuous optimized delay using 2X to 5X fewer cells.

5 Concluding Remarks

The joint gate sizing/library optimization method presented in this paper provides a new tool for realizing size optimized logic circuits that complements and enhances existing discrete as well as continuous sizing methods. If a given network (or suite of networks) is to be discrete size optimized, a designer now has a way to find the best set of cells to include (or ask for) in a library. If a given network is to be continuous size optimized, a designer now has a way to significantly reduce the number of unique cell sizes required without adversely affecting optimized circuit performance.

References

[1] D. Marple, Performance optimization of digital VLSI circuits, Tech. Report: CSL-TR-86-308, Computer Systems Laboratory, Stanford University, Oct. 1986.
 [2] K. Lowe and P.G. Gulak, Gate sizing and buffer insertion for optimizing performance in power constrained BiCMOS circuits, Proc. IEEE Int. Conf. Computer-Aided Design, 1993, pp. 216-219.

[3] P. Chan, Algorithms for library-specific sizing of combinational logic, Proc.DAC, June 1990, pp. 353-356.
 [4] S. Lin, M. Marek-Sadowska and E. Kuh, Delay and area optimization in standard-cell design, Proc. DAC, June 1990, pp. 349-352.
 [5] U. Hinsberger and R. Rolla, Cell based performance optimization of fanout-free trees, IEEE Trans. on Comp. Aided Design, vol. 11, no. 10, Oct. 1992, pp. 1371-1321
 [6] P. Duchene and M. Declercq, Strategies for CMOS/BiCMOS gate usage on sea-of-gates arrays. Proc. IEEE CICC, May 1991, pp. 14.2.1-14.2.4.
 [7] J. Ecker, Geometric programming: methods, computations and applications, Siam Review, vol. 22, no. 3, July 1980, pp. 338-362.
 [8] J. Fishburn and A. Dunlop, TILOS: a posynomial programming approach to transistor sizing, Proc. IEEE Int. Conf. Computer-Aided Design, 1985, pp. 326-328.
 [9] G. Vanderplaats and H. Sugimoto, A general-purpose optimization program for engineering design, Computer & Structures, vol. 24, no. 1, 1986, pp. 13-21.
 [10] J. Fishburn, A depth-decreasing heuristic for combinational logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between, Proc. DAC, June 1990, pp. 361-364.
 [11] R. Hadaway, et al., A sub-micron BiCMOS technology for telecommunications, Microelectronic Engineering, Elsevier, vol. 15, 1991, pp. 513-516.
 [12] F. Najm, Transition density: a new measure of activity in digital circuits, IEEE Trans. on Comp. Aided Design, vol. 12, no. 2, Feb. 1993, pp. 310-323.