ConvexFit: An Optimal Minimum-Error Convex Fitting and Smoothing Algorithm with Application to Gate-Sizing

Sanghamitra Roy, Weijen Chen and Charlie Chung-Ping Chen
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
1415 Engineering Drive, Madison,WI-53706
{*roy1@, weijenchen@, chen@engr*}*.wisc.edu*

ABSTRACT

Convex optimization has gained popularity due to its capability to reach global optimum in a reasonable amount of time. Convexity is often ensured by fitting the table data into analytically convex forms such as posynomials. However, fitting the look-up tables into the posynomial forms with minimum error itself may not be a convex optimization problem and hence excessive fitting errors may be introduced.

In this paper, we propose to directly adjust the look-up table values into a numerically convex look-up table without explicit analytical form. We show that numerically "convexifying" the table data with minimum perturbation can be formulated as a convex semidefinite optimization problem and hence optimality can be reached in polynomial time. Without an explicit form limitation, we find that the fitting error is significantly reduced while the convexity is still ensured. As a result, convex optimization algorithms can still be applied. Furthermore, we also develop a "smoothing" algorithm to make the table data smooth and convex to facilitate the optimization process.

Results from extensive experiments on industrial cell libraries demonstrate that our method reduces 30X fitting error over a well-developed posynomial fitting algorithm. Its application to circuit tuning is also presented.

1. INTRODUCTION

Convex optimization has gained popularity in the VLSI circuit optimization society due to its capability of reaching global optimality for large scale optimization applications [2], [5], [6], [7].

To ensure the convexity of a given optimization problem with given table data such as gate delay with respect to the gate sizes and input slews, significant fitting efforts are required to obtain an analytically explicit convex functional form such as posynomials to closely represent the look-up table data with minimum error [2], [5].

Unfortunately, the fitting process to posynomials with unknown exponents may *not* be a convex optimization problem and is often trapped in the local minimum except for the simple monomial case. Although systematic fitting methods such as K-mean algorithms [12] have been proposed to reduce the fitting errors, the optimality is still not guaranteed. As a result, the fitting error may be too excessive or require human intervention for practical applications. Although generalized posynomial form [2] has been proposed, this issue is still not solved.

Alternatively, we can directly use the look-up table data which can be generated experimentally (for example by running SPICE simulation) [1]. Using finite difference method, we can still obtain sensitivity and even hessian which are sufficient for optimization usage. However, as pointed out in [2], this method is not capable of ensuring convexity required to ensure quick global convergence. As a result, it is necessary to modify the data such that both convex and smooth properties can be guaranteed.

In this paper, we propose to directly adjust the look-up table data values into a numerically convex table without explicit analytical form. We show that numerically "convexifying" the look-up table data with minimum perturbation can be formulated as a convex semidefinite optimization problem and hence its optimality can be reached in polynomial time. Without an explicit form limitation, we find that the fitting error is significantly reduced while the convexity is still ensured. As a result, convex optimization algorithms can still be applied. Furthermore, we also develop a "smoothing" algorithm to make the table data smooth and convex to facilitate the optimization process.

Results from extensive experiments on a full industrial cell library demonstrate that our method reduces 30X fitting error over a well-developed posynomial fitting algorithm. We also show that the eigenvalue distributions of the hessian are much better preserved compared to posynomial-based fitting algorithm. At this point it will be useful to mention that the terms like 'AN2', 'ANB2', 'INV', 'NR2' that will be used at several places in the paper refer to cell names from the standard cell library used in our experiments.

The application of convex optimization to VLSI is numerous. The most widely used is the gate sizing problem where the gate delay is reasonably close to the convex function of gate size and gate load. In our experimental result section, we also demonstrate the application of our algorithm to the large scale circuit tuning problem.

The organization of the paper is as follows. In Section 2 we provide some general background on convexity, posynomials and semidefinite programming. We describe our convex model and our problem formulation in section 3.1. We present our "smoothing" algorithm in section 3.2. In section 4 we provide experimental results of $ConvexFit$ on industrial cell libraries. We also compare our results with $PosynomialFit$, our posynomial modelling technique. We conclude our discussion in section 5.

2. FUNDAMENTAL CONCEPTS

In this section, the fundamental theorem about convexity and semidefinite programming is introduced.

2.1 Convexity, Posynomial, and Hessian

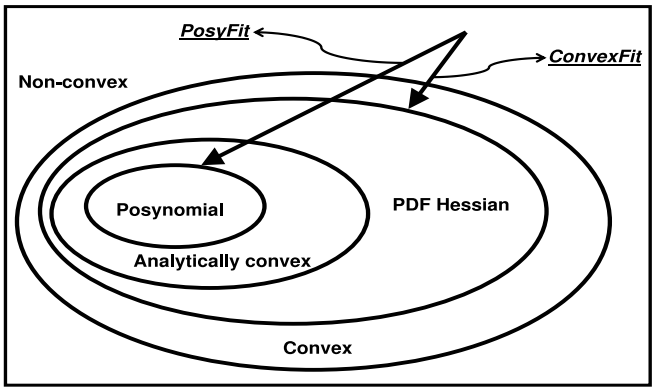

Figure 1: Convexity Taxonomy and "Convexifying" process

Since convex representation of data is important for optimization problems based on convex programming techniques, we now introduce the fundamental definition of convexity. A function $f(x)$ is convex if

$$f(\,(1-\lambda)a+\lambda b) \leq (1-\lambda)f(a)+\lambda f(b)$$
$$\forall a,b \in DOMf, and\, \forall \lambda \in (0,1)$$

If $f(x)$ is 2nd-order differentiable then $f(x)$ is convex if and only if $\nabla^2 f(x) \succeq 0$ for all $x \in DOMf$, where $\nabla^2 f(x)$ is the Hessian of $f(x)$, denoted as $H(x)$ and is defined as

$$[H(x)]_{ij} = [\nabla^2 f(x)]_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}, i,j = 1 \ldots n$$

and $\nabla^2 f(x) \succeq 0$ means the Hessian of $f(x)$ is positive semidefinite, i.e. all the eigenvalues of the Hessian are greater or equal to zero.

A posynomial is a function almost like a polynomial but with positive coefficients and real exponents. The definition of a posynomial is

$$f(x) = \sum_{j=1}^{k} c_j \prod_{i=1}^{n} {x_i}^{\alpha_{ij}},$$

where c_j, $j = 1 \ldots n$ are positive real numbers, and α_{ij} are real numbers. It is well known that posynomial is convex after the exponential transformation $e^y = x$.

Note that posynomial is only a proper-subset of analytically convex functions. For example, $\frac{1}{x+1}$ or $-log(x)$ are also convex functions but not in posynomial form. The more general the convex form, the better it can be used for convex fitting. Recently, Kishore Kasamsetty et al expanded the posynomial form to generalized posynomial [2], $G_k(x) = . \sum_{j=1}^{k} c_j \prod_{i=1}^{n} G_{k-1}^{\alpha_{ij}}$, where $\alpha_{ij} \geq 0$ and $G_0(x)$ is a posynomial function.

2.2 Semidefinite Programming

A semidefinite program (SDP) is an optimization problem of the form:

$$\begin{aligned} \text{SDP: minimize} \quad & C \bullet X \\ \text{s.t.} \quad & A_i \bullet X = b_i \text{ ,i=1,...,m,} \\ & X \succeq 0\,, \end{aligned}$$

The objective function is the linear function $C \bullet X$ and there are m linear equations that X must satisfy, namely $A_i \bullet X = b_i\,, i = 1, ..., m$. The variable X also must lie in the (closed convex) cone of positive semidefinite symmetric matrices. If $C(X)$ is a linear function of X, then $C(X)$ can be written as $C \bullet X$, where

$$C \bullet X = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij}. \tag{1}$$

We now introduce the primal form (P) and the dual form (D) of SDP which we use in our optimization problem :

$$\begin{aligned} (P) \quad & minimize \quad \sum_{j=1}^{n_b} \langle C_j, X_j \rangle \\ & subject\ to \quad \sum_{j=1}^{n_b} \langle A_{i,j}, X_j \rangle = b_i, \\ & \qquad i = 1, ..., m,\ X_j \in K_j \qquad (2) \\ (D) \quad & maximize \quad \sum_{i=1}^{m} b_i y_i \\ & subject\ to \quad \sum_{i=1}^{m} A_{i,j} y_i + S_j = C_j, \\ & \qquad j = 1, ..., n_b,\ S_j \in K_j \qquad (3) \end{aligned}$$

where each cone K_j is a set of symmetric positive semidefinite matrices. Hence,

$$\begin{aligned} S_j &\succeq 0, \\ \textstyle\sum_{i=1}^{m} A_{i,j} y_i - C_j &\preceq 0, \\ \textstyle\sum_{i=1}^{m} A_{i,j} y_i &\preceq C_j \end{aligned} \tag{4}$$

3. CONVEX FITTING PROBLEM FORMULATION

In this section, we introduce the formulation of minimum-error convex fitting problem and its solution.

3.1 Convex Fitting Problem Formulation

Fundamentally, we are interested in knowing what are the most general convex functions. If such a general function form exists, is there any way we can fit general functions especially in table form to this function with guaranteed minimum error? We now describe the minimum-error convex fitting problem formally.

Given an analytical or numerical function $g(\mathbf{x})$, we define the minimum-error convex fitting problem as follows:

$$\begin{aligned} ConvexFit: \\ & minimize \quad \sum_{\mathbf{x_m} \in DOMg} | \, f(\mathbf{x_m}) - g(\mathbf{x_m}) \, | \\ & subject\ to \quad f(\mathbf{x_m})\, is\, convex,\ \mathbf{x_m} \in DOMg \end{aligned}$$

Let $f(\mathbf{x_m}) - g(\mathbf{x_m}) = \delta'(\mathbf{x_m})$, then $\delta'(\mathbf{x_m})$ is the perturbation of $g(\mathbf{x_m})$, the task of $ConvexFit$ is to minimize the

perturbation of $g(\mathbf{x})$ to make the hessian of $f(\mathbf{x})$ convex. We rewrite the problem formulation as follows:

$$
\begin{aligned}
ConvexFit' : \quad & \\
& minimize \quad \sum_{\mathbf{x_m} \in DOMg} \mid \delta'(\mathbf{x_m}) \mid \\
& subject\ to \quad \nabla^2(g(\mathbf{x_m}) + \delta'(\mathbf{x_m})) \succeq 0, \\
& \qquad \mathbf{x_m} \in DOMg
\end{aligned}
$$

Since $\mid \delta(\mathbf{x_m}) \mid$ is not a linear function of $\mathbf{x_m}$, we use $-\delta(\mathbf{x_m}) \leq \delta'(\mathbf{x_m}) \leq \delta(\mathbf{x_m})$, where $\delta(\mathbf{x_m}) \geq 0$ to represent it. Under this transformation, we get the following formulation:

$$
\begin{aligned}
ConvexFit'' : \quad & \\
& minimize \quad \sum_{\mathbf{x_m} \in DOMg} \delta(\mathbf{x_m}) \\
& subject\ to \quad \nabla^2(g(\mathbf{x_m}) + \delta'(\mathbf{x_m})) \succeq 0, \\
& \qquad -\delta(\mathbf{x_m}) \leq \delta'(\mathbf{x_m}) \leq \delta(\mathbf{x_m}), \\
& \qquad \delta(\mathbf{x_m}) \geq 0, \\
& \qquad \mathbf{x_m} \in DOMg \qquad (5)
\end{aligned}
$$

It is easy to see that the $ConvexFit''$ formulation can be easily fitted into the dual form (D) of semidefinite programming framework [3]. Since semidefinite programming is a convex optimization problem, the $ConvexFit''$ problem can be optimally solved by any semidefinite programming solver.

Since the domain of interests of $g(\mathbf{x})$, or $DOMg(\mathbf{x})$, is often finite, we can use a finite difference scheme to approximate the sensitivity and hessian of $g(\mathbf{x})$ by $\frac{\partial g(\mathbf{x})}{\partial x_i} \cong \frac{g(\mathbf{x}+\Delta\,\mathbf{e_i})-g(\mathbf{x}-\Delta\,\mathbf{e_i})}{2\Delta}$ and $[\nabla^2 g(\mathbf{x})]_{ij} = \frac{\partial^2 g(\mathbf{x})}{\partial x_i \partial x_j} \cong \frac{g(\mathbf{x}+\Delta\,\mathbf{e_i}+\Delta\,\mathbf{e_j})-g(\mathbf{x}-\Delta\,\mathbf{e_i}+\Delta\,\mathbf{e_j})-g(\mathbf{x}+\Delta\,\mathbf{e_i}-\Delta\,\mathbf{e_j})+g(\mathbf{x}-\Delta\,\mathbf{e_i}-\Delta\,\mathbf{e_j})}{4\Delta\Delta}$, where $\mathbf{e_i}$ is a vector with one in i^{th} entry and zero in others.

3.2 Smoothing the ConvexFit Model

The convex optimizer will converge faster if the sensitivity and even hessian are continuous. Let f be a discrete function of $\mathbf{x} = [x_1, x_2, ..., x_n]^T$. We use the following quadratic form to generate smooth data values of f at intermediate points $\mathbf{x'} = \mathbf{x} + \Delta\mathbf{x}$.

$$f(\mathbf{x} + \Delta\mathbf{x}) = \frac{1}{2}\Delta\mathbf{x}^T H_f(\mathbf{x})\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{b} + C$$

where $C = f(\mathbf{x})$, $\mathbf{b} = [\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, ..., \frac{\partial f(\mathbf{x})}{\partial x_n}]^T$ and H_f is the Hessian of f. In the n-dimensional case, we can have 2^n surrounding data points for an intermediate point $\mathbf{x'}$. We calculate the value at $\mathbf{x'}$ by using the quadratic form from each surrounding point. We then find the smoothed value at $\mathbf{x'}$ by a weighted sum of the values obtained from the 2^n points.

An n-dimensional space is divided into hypercubes. Each point $(x_1, x_2, .., x_n)$ not on the boundary of the hypercubes, has 2^n surrounding points. Our smoothing algorithm uses the surrounding 2^n points to approximate the value at point $(x_1, x_2, .., x_n)$. These points are labeled as $(x_{1l}, x_{2l}, ...)$, $(x_{1u}, x_{2l}, ...), (x_{1l}, x_{2u}, ...), (x_{1u}, x_{2u}, ...)$.... where $x_{1l} < x_1 < x_{1u}$, $x_{2l} < x_2 < x_{2u}, ...$ and so on.

We have to choose the weight associated with each point such that our curve can pass through the original data and is also continuously differentiable everywhere. We do not directly use $1/distance$ as the weight of these points. This may result in non-continuity at the boundary of each cube. We calculate weight $x_{1w} = (x_{1u} - x_1)/(x_{1u} - x_{1l})$. Similarly, we can calculate $x_{2w}, x_{3w}, ...x_{nw}$. We use $(x_{1w} \times x_{2w} \times x_{3w} ... \times x_{nw})$ as the weight of point $(x_{1l}, x_{2l}, x_{3l}, ..., x_{nl})$. Similarly we use $((1 - x_{1w}) \times x_{2w} \times x_{3w} ... \times x_{nw})$ as weight of point $(x_{1u}, x_{2l}, x_{3l}, ..., x_{nl})$ and so on. Our smoothed curve passes through the original points. Hence smoothing does not add additional error over $ConvexFit$.

The sum of the weights of all the surrounding points is one. We use $\sum(value \times weight)$ as our smoothed value of point $(x_1, x_2, .., x_n)$. Following a similar methodology we can deal with the points in the boundary. Figure 2 illustrates the weight calculation for 3-dimensional space.

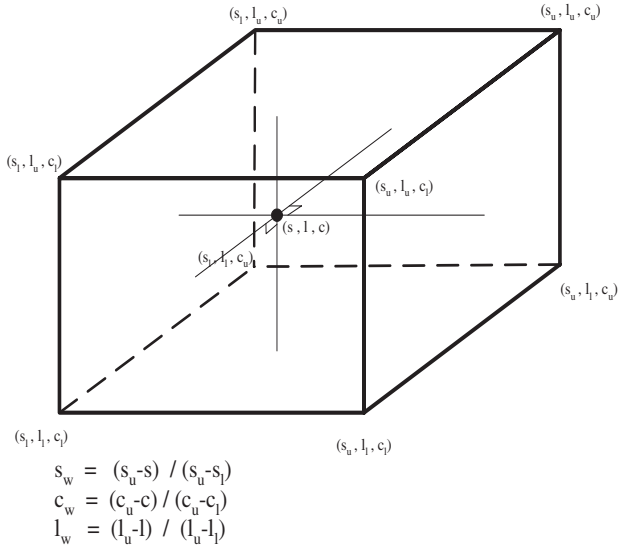

Figure 2: Smoothing for 3-dimension: n = 3, x1 = slew, x2 = load, x3 = input capacitance

3.3 Posynomial Fitting Procedure for comparison

We initially modelled cell data as posynomials due to the popularity of geometric programming. While searching reliable and automatic posynomial fitting procedures, we found that it was extremely difficult to find universally accurate posynomials for even just the delay or transition time which naturally should already be very close to polynomial forms. We tried several methods which actually worked for many cases while were not suitable for many other cases. We now introduce those methods.

The posynomial modelling procedure is essentially done via least-square regression analysis on the cell data. The posynomial parametric regression problem can be formally defined as follows:

$$
\begin{aligned}
PosynomialFit : \quad & minimize \quad \sum_{m=1}^{z} ((\sum_{j=1}^{k_m} c_j \prod_{i=1}^{n} {x_{mi}}^{\alpha_{ij}}) - b_m)^2 \\
& subject\ to \quad c_j \geq 0 \qquad (6)
\end{aligned}
$$

where z is the number of sets of tunable parameters , n is

the number of tunable parameters which affect the metric being approximated, $x_{mi} \in \Re$ is the i_{th} entry of the m_{th} set of tunable parameters, $b_m \in \Re$ is one of z different values from the cell look-up table each corresponding to the m_{th} set of tunable parameters. k_m, c_j, and α_{ij} are the unknown parameters we are trying to determine.

If the exponents, α_{ij}, are already known, the problem of $PosynomialFit$ is a convex least square fitting problem which can be optimally solved by any convex solver. The issue is that when the exponents are unknown, the problem is not in any known convex form. Therefore, the convexity of the problem is not guaranteed. There is, however, one special case. When there is only one term in the posynomial, it degenerates into monomial form which can be solved optimally. This can be seen by taking a logarithm on both sides of the equation.

We developed three different algorithms for posynomial modelling.

Approach 1 - Posynomial Characterization with Fixed Exponents and Unknown Coefficients : Select and fix a number of monomial terms to use in the characterization process. Select and fix a set of exponent values to use for each variable in the posynomial expression. Determine the best coefficients using least-square fitting.

Approach 2 - 1-Phase Posynomial Characterization with unknown Exponents and Unknown Coefficients: Select and fix a number of monomial terms to use in the characterization process. Determine the unknown coefficients and unknown exponents for the posynomial expression using least-square fitting. The following is an example of a posynomial for cell AN2 obtained by using this approach: cell_rise posynomial for $pin = I1$ is:

$$\begin{aligned} &0.00014 \times (slew)^{0.0049} \times (load)^{0.8783} \times (cap)^{-1.7592} \\ &+0.00598 \times (slew)^{-0.3466} \times (load)^{-0.5262} \times (cap)^{0.6315} \\ &+0.51642 \times (slew)^{0.6948} \times (load)^{2.1138} \times (cap)^{2.4264} \\ &+6.0844 \times (slew)^{0.3135} \times (load)^{-0.0656} \times (cap)^{0.8093} \\ &+8.18 \times 10^{-10} \times (slew)^{2.4264} \times (load)^{-0.2173} \times (cap)^{-1.4685} \end{aligned}$$

Approach 3 - 2-Phase Posynomial Characterization with unknown Exponents and Unknown Coefficients: (Similar to K-Mean Algorithm [12]) We explain this in the context of our cell delay which depends on the drive strength, input slew rate and the load capacitance of the cell. Phase 1: For each drive strength instance of a particular cell, individually generate an accurate posynomial expression based on only input transition and output load capacitance (with unknown coefficients and unknown exponents for the posynomial expression solved using least-square fitting). Phase 2: Aggregate all results from phase 1 together, and add to each term a new posynomial term expressed as a function of the input capacitance (which is directly related to the transistor width drive strength)

Table 1 summarizes the results of these approaches for cells AN2 and ANB2. SE is the total square error for the cell delay model. The percentage of total data points falling within ±20% fitting error is also shown in the table.

4. EXPERIMENTAL RESULTS ON INDUSTRIAL CELL LIBRARY

Table 1: Posynomial modelling approaches

Std. cell	Approach 1 SE (ns^2)	Approach 1 pts within 20% error	Approach 2 SE (ns^2)	Approach 2 pts within 20% error	Approach 3 SE (ns^2)	Approach 3 pts within 20% error
AN2	4.34	43%	0.88	96%	0.89	94%
ANB2	7.43	12%	4.26	50%	4.95	47%

In this section, we first introduce how to apply $ConvexFit$ to a popular industrial cell library to generate convex delay and transition time models. Next, we present the experimental comparison between $PosynomialFit$ and $ConvexFit$ algorithms. Then we introduce the results obtained from our smoothing algorithm. Finally, we illustrate the application of our algorithm to gate-sizing.

4.1 Applying ConvexFit in generating standard cell delay models

Our $ConvexFit$ formulation (5) can be easily applied to model standard cell delay look-up tables. Let us denote our lookup table, cell rise time as $f(s, c, l)$, a function of input slew s, input capacitance c, and output load l. We have to introduce minimum perturbation in the cell rise values to make them purely convex. The perturbed function obtained by running $ConvexFit$ is our generated delay model for the standard cells.

Table 2: Cell-Rise Fitting Errors Comparison

Cell name	PosynomialFit SE $(ns)^2$	PosynomialFit AE (ns)	ConvexFit SE $(ns)^2$	ConvexFit AE (ns)
AN2	0.883	0.035	0.333	0.0113
AOI222	9.214	0.137	0.143	0.0079
BUF	3.861	0.047	0.679	0.0125
INV	9.427	0.087	0.052	0.0044
MAOI1	8.100	0.125	0.0445	0.0037
MUX4	5.900	0.108	0.031	0.0064
ND3	1.774	0.068	0.236	0.0143
OA12	3.523	0.084	0.369	0.0109
OR3B2	4.009	0.087	0.074	0.0041
XOR2	2.537	0.064	0.000	0.0002

4.2 Comparison of PosynomialFit and ConvexFit

We now present the experimental results of $ConvexFit$ and $PosynomialFit$ on a real industrial cell library. It is a $0.13\mu m$ family standard cell library containing 415 generic core cells and 53 I/O cells. We performed our experiments using 67 combinational cells from this library. We have used the DSDP5.7 [3] solver in C to run our semidefinite optimization $ConvexFit$ [16]. $PosynomialFit$ (using approach 2 in subsection 3.3) was implemented in C++ using the CFSQP solver [4]. All experiments were performed on a PC with 1.60GHz Microprocessor, 256 MB RAM and 60 GB hard drive running Windows XP. Experiments were performed for cell-rise, cell-fall, rise-transition and fall-transition look-up tables for each cell, and for one input pin per cell. Tables 2, 3, 4, 5 summarize the total square error(SE) and the average absolute error(AE) for the four different look-up tables(results for only ten cells are shown due to limitation of space). SE is calculated by summing the square of the error for each data point in the look-up table. AE is calcu-

lated by summing the absolute error for each data point in the look-up table, and then dividing by the total number of data points. It can be observed that *ConvexFit* shows more than 30X reduction in fitting error over *PosynomialFit*. *ConvexFit* has an average square error of 0.26, while average square error of *PosynomialFit* is 9.5307. Figures 3, 4, 5, 6 showing the error distribution for the two techniques demonstrate the significant reduction in fitting error of *ConvexFit* over *PosynomialFit*. Table 6 presents the runtime comparisons. It shows that *ConvexFit* is at least 7-8 X faster than *PosynomialFit*.

Table 3: Cell-Fall Fitting Errors Comparison

Cell name	PosynomialFit		ConvexFit	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.130	0.0130	0.056	0.0046
AOI222	1.689	0.0637	0.071	0.0067
BUF	0.524	0.0189	0.053	0.0029
INV	1.768	0.0391	0.032	0.0046
MAOI1	1.017	0.0483	0.073	0.0071
MUX4	0.926	0.0462	0.062	0.0061
ND3	1.506	0.0672	0.038	0.0078
OA12	0.506	0.0289	0.050	0.0036
OR3B2	1.274	0.0530	0.003	0.0009
XOR2	0.395	0.0292	0.000	0.000

Table 4: Rise-Transition Fitting Errors Comparison

Cell name	PosynomialFit		ConvexFit	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	4.818	0.0832	1.800	0.0276
AOI222	52.601	0.3521	0.711	0.0194
BUF	18.392	0.0990	3.378	0.0267
INV	42.715	0.1843	0.260	0.0094
MAOI1	39.820	0.2859	1.140	0.0286
MUX4	33.874	0.2786	0.177	0.0095
ND3	9.561	0.1526	0.127	0.0076
OA12	18.054	0.1689	2.024	0.0292
OR3B2	22.853	0.2164	0.655	0.0241
XOR2	16.216	0.1695	0.002	0.0006

Table 5: Fall-Transition Fitting Errors Comparison

Cell name	PosynomialFit		ConvexFit	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.854	0.0421	0.207	0.0092
AOI222	9.531	0.1582	0.048	0.0056
BUF	2.048	0.0355	0.361	0.0083
INV	5.317	0.0664	0.035	0.0033
MAOI1	5.218	0.1081	0.217	0.0123
MUX4	4.306	0.0933	0.026	0.0039
ND3	6.414	0.1329	0.000	0.0001
OA12	2.237	0.0576	0.224	0.0088
OR3B2	7.531	0.1242	0.389	0.0169
XOR2	2.943	0.0690	0.000	0.0000

Figures 7 and 8 show the cell-rise, cell-fall look-up tables $f(s, c, l)$ for AN2, the corresponding convex and posynomial model data mapped to a single dimension to illustrate the fitting efficiency of these techniques. These figures show only a small subset (zoomed) of the total data points in the look-up table.They show that *ConvexFit* consistently performs better than *PosynomialFit* throughout the range of data.

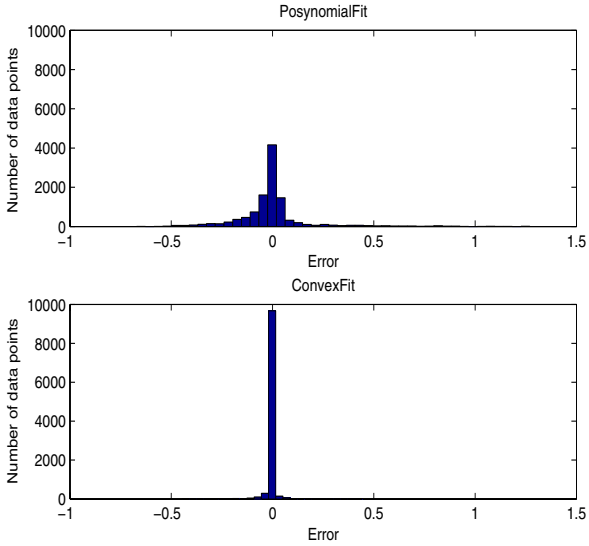

Figure 3: Cell-Rise error distribution for PosynomialFit and ConvexFit

The positive definiteness of our model is indicated by the eigenvalues of its Hessian. The real part of all the eigenvalues must be positive to ensure convexity. *ConvexFit* will make the eigenvalues positive using minimum perturbation to the original data. We plot the eigen-distribution of Hessian for cell AN2 in figures 9 and 10. It can be easily observed that *ConvexFit* preserves most of the positive eigenvalues while moving the negative eigenvalues to be just positive. As for the result generated from *PosynomialFit*, although all the eigenvalues are made positive, the whole distributions are disturbed.

Table 6: Execution times

Mode	PosynomialFit (sec)	ConvexFit (sec)
Cell-Rise	952	125
Cell-Fall	1063	121
Rise-Transition	1019	178
Fall-Transition	995	131

4.3 Smoothing of ConvexFit for standard cells

Our *ConvexFit* models for standard cells were smoothed according to the technique mentioned in subsection 3.2. As the delay depends on 3 parameters (s, c, l) we used weighted sum of the values obtained from the 8 $(= 2^3)$ surrounding points. Figures 11 and 12 give a visual illustration of the original cell-rise data and the convexified and smoothed model for cells NR2 and INV.

4.4 Gate sizing and delay calculations

We test the accuracy of our model in the gate sizing optimization problem. We performed the experiments using our gate sizing tool Glaive implemented in C++. We conducted our experiments on ISCAS85 benchmark circuits where the number of gates ranged from 214 to 3512. The synthesized ISCAS85 design using the standard cell library was used as

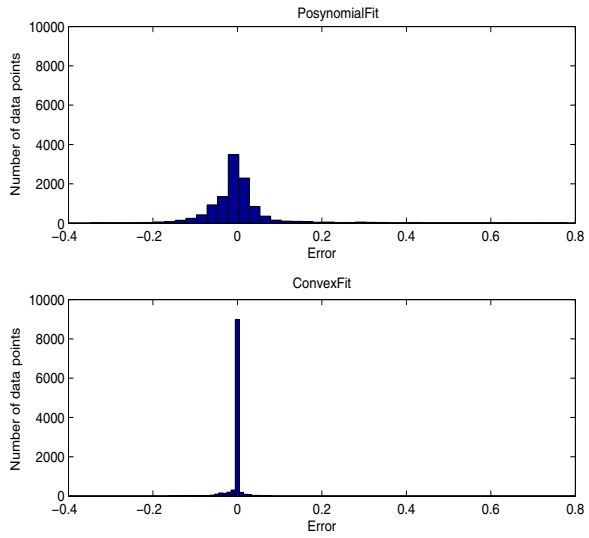

Figure 4: Cell-Fall error distribution for PosynomialFit and ConvexFit

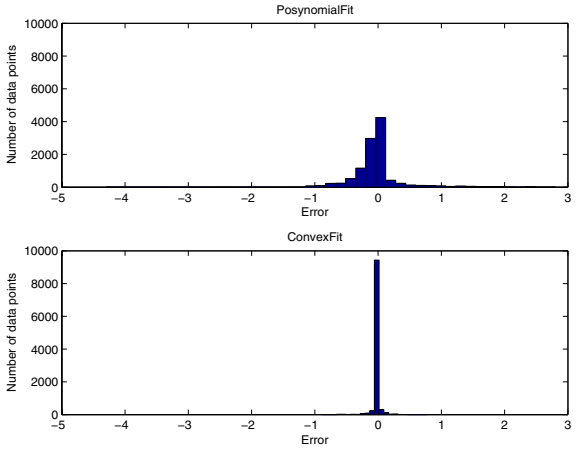

Figure 5: Rise-Transition error distribution for PosynomialFit and ConvexFit

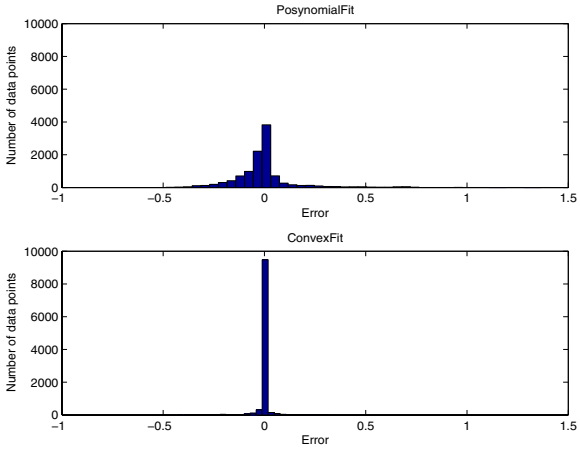

Figure 6: Fall-Transition error distribution for PosynomialFit and ConvexFit

input to our gate sizing software. We used the arrival time at the sink node of the netlist as our objective function. Table 7 summarizes the delay for various benchmark circuits using $PosynomialFit$ and $ConvexFit$ models in our gate sizing tool. We have also compared our delay with that for the same synthesized designs, from a popular commercial synthesis tool. All delay measurements were done using Synopsys Primetime. It can be observed that both of our models perform better than the optimized delay from the synthesis tool. $ConvexFit$ shows on average 1.18% improvement in delay over $PosynomialFit$. $PosynomialFit$ might rarely show a marginally better delay than $ConvexFit$. This can be explained by Figures 7, 8 where in certain smaller sections of the curve, $PosynomialFit$ might give a closer fitting than $ConvexFit$.

Table 7: Results of sizing various circuits

circuit	Commercial Synthesis tool Delay (ns)	Gate sizing result		Delay ratio CFit/PFit
		PosynomialFit Delay (ns)	ConvexFit Delay (ns)	
C432	2.82	2.66	2.67	1.004
C499	1.80	1.71	1.69	0.989
C880	2.45	2.13	2.11	0.991
C1355	2.85	2.18	2.13	0.977
C1908	2.74	2.32	2.26	0.974
C5315	2.48	2.22	2.19	0.986
C6288	11.26	10.02	10.01	0.999

5. CONCLUSION

Convex optimization has gained popularity due to its capability to reach global optimum in a reasonable amount of time. Convexity is often ensured by fitting the table data into analytically convex forms such as posynomials. However, fitting the look-up tables into the posynomial forms with minimum error itself may not be a convex optimization problem and hence excessive fitting errors may be introduced. In this paper, we propose to directly adjust the look-up table values into a numerically convex look-up table without explicit analytical form. We show that numerically "convexifying" the look-up table data with minimum perturbation can be formulated as a convex semidefinite optimization problem and hence optimality can be reached in polynomial time. We demonstrate over 30X improvement in fitting error over a well-developed posynomial fitting procedure. We also develop a "smoothing" algorithm to further make the table data smooth and convex to facilitate optimization process. We illustrate the effectiveness of this model in a convex optimization problem by providing extensive results for using our model in the optimal gate-sizing of standard cells.

6. ACKNOWLEDGEMENTS

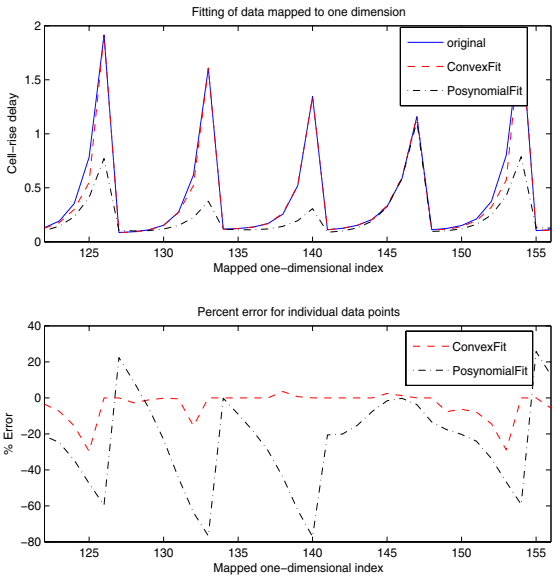

Figure 7: Fitting of PosynomialFit and ConvexFit for data mapped to one dimension:Cell-Rise

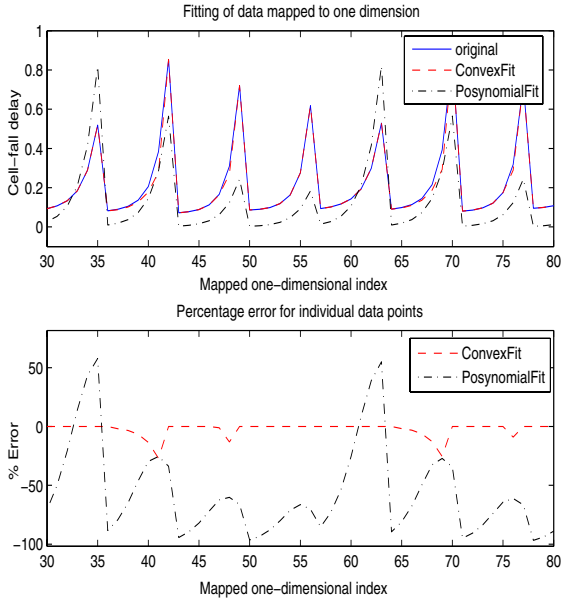

Figure 8: Fitting of PosynomialFit and ConvexFit for data mapped to one dimension: Cell-Fall

This work was partially funded by Intel, TSMC, UMC, Faraday, SpringSoft, National Science Foundation under grants CCR- 0093309, CCR-0204468 and National Science Council of Taiwan, R.O.C. under grant NSC 92-2218-E-002-030.

7. REFERENCES

[1] V.B.Rao, T.N.Trick, and I.N.Hajj, "A table-driven delay-operator approach to timing simulation of MOS VLSI circuits", in *Proceedings of the 1983 International Conference on Computer Design*, pp.445-448, 1983.

[2] Kishore Kasamsetty, Mahesh Ketkar and Sachin S. Sapatnekar, "A New Class of Convex Functions for Delay Modeling and their Application to the Transistor Sizing Problem", in *IEEE Journal of Solid-State Circuits*, Vol. 37, pp. 521-525, Apr.2002.

[3] Steven J. Benson and Yinyu Ye, "DSDP5 User Guide - The Dual-Scaling Algorithm for Semidefinite Programming", *Technical Report ANL/MCS-TM-255*, February 16, 2005.

[4] C. Lawrence, J.L Zhou and A.L Tits, "User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints", *Technical Report TR-94-16rl*, April, 1997.

[5] J. Fishburn and A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing", in *Proceedings of the IEEE International Conference on Computer-Aided Design*,pp.326-328, 1985.

[6] 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", in *IEEE Transaction on Computer-Aided Design*,vol. 12,pp.1621-1634, 1993.

[7] H. Tennakoon and C. Sechen, "Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step", in *International Conference on Computer-Aided Design*,pp.395-402, 2002.

[8] M. Vujkovic and C. Sechen, "Optimized power-delay curve generation for standard cell ICs", in *International Conference on Computer-Aided Design*,pp. 387-394, 2002.

[9] S. Boyd and L. Vandenberghe, "Convex Optimization", Cambridge University Press, 2003. J.-M. Shyu, A. L. Sangiovanni-Vincentelli,

[10] J.M Shyu, A.L Sangiovanni, J.Fishburn, and A.Dunlop, "Optimization-based transistor sizing", in *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 400-409, Apr 1988.

[11] N.P Jouppi,"Timing analysis and performance improvement of MOS VLSI design," in *IEEE Transactions on Computer-Aided Design*,vol. CAD6, pp.650-665, July 1987

[12] S.Z Selim and M.A Ismail,"K-Means-Type algorithms: a generalized convergence theorem and characterization of local optimality," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp.81-87, January 1984.

[13] Chung-Ping Chen, Chris C. N. Chu, and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits And Systems*, vol. 18, No. 7, pp. 1014-1025, July 1999.

[14] D. Sinha and Hai Zhou, "Gate Sizing for Crosstalk Reduction under Timing Constraints by Lagrangian Relaxation," in *Proceedings of the International Conference on Computer Aided Design*, pp. 14-19, 2004.

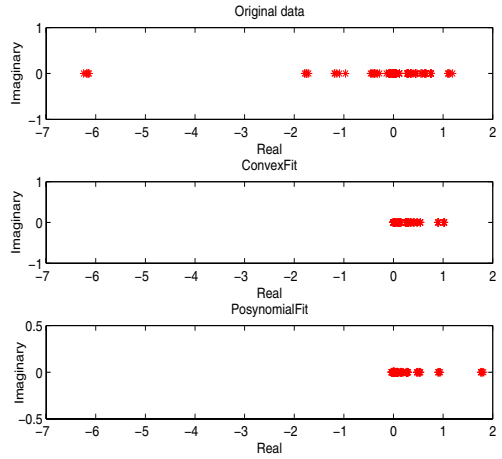

Figure 9: Eigenvalues distribution in the complex plane

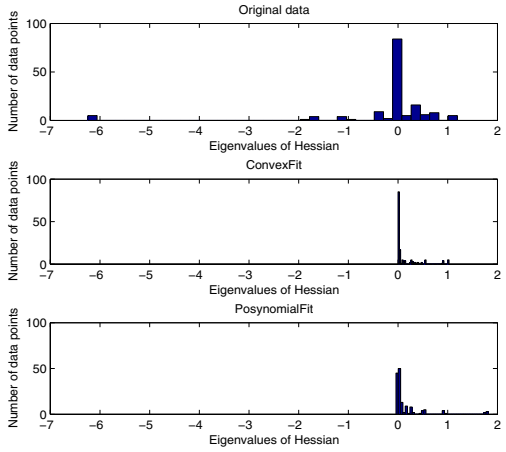

Figure 10: Eigenvalues distribution statistics and ConvexFit

[15] L. Vandenberghe and S. Boyd,"Applications of semidefinite programming", in *Applied Numerical Mathematics*, 29:283-299, 1999.

[16] Sanghamitra Roy, Weijen Chen and Charlie Chung-Ping Chen,"ConvexFit: an optimal minimum error convex fitting tool", http://vlsi.ece.wisc.edu/Tools.htm, July 2005.

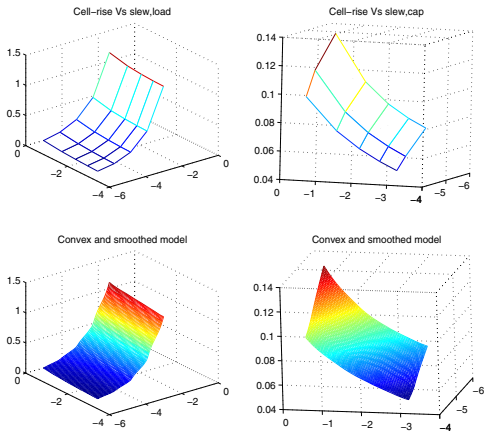

Figure 11: Plot of original data and Smooth ConvexFit model for cell NR2

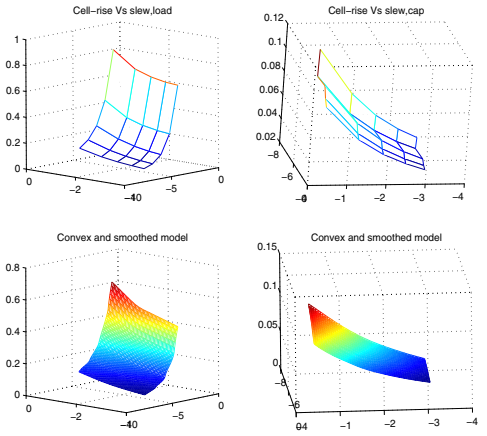

Figure 12: Plot of original data and Smooth ConvexFit model for cell INV