ConvexSmooth: A simultaneous convex fitting and smoothing algorithm for convex optimization problems

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

ABSTRACT

Convex optimization problems are very popular in the VLSI design society due to their guaranteed convergence to a global optimal point. Table data is often fitted into analytical forms like posynomials to make them convex. However, fitting the look-up tables into posynomial forms with minimum error itself may not be a convex optimization problem and hence excessive fitting errors may be introduced. In recent literature numerically convex tables have been proposed. These tables are created optimally by minimizing the perturbation of data to make them numerically convex. But since these tables are numerical, it is extremely important to make the table data smooth, and yet preserve its convexity. Smoothness will ensure that the convex optimizer behaves in a predictable way and converges quickly to the global optimal point.

In this paper, we propose to simultaneously create optimal numerically convex look-up tables and guarantee smoothness in the data. We show that numerically "convexifying" and "smoothing" the table data with minimum perturbation can be formulated as a convex semidefinite optimization problem and hence optimality can be reached in polynomial time.

We present our convexifying and smoothing results on industrial cell libraries. ConvexSmooth shows 14X reduction in fitting error over a well-developed posynomial fitting algorithm.

1. INTRODUCTION

Convex optimization problems are very popular in the VLSI design society due to their guaranteed convergence to a global optimal point[2], [5], [6], [7]. While optimizing tabular data such as gate delay, 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]. However, fitting the look-up tables into posynomial forms with minimum error itself may not be a convex optimization problem and hence excessive fitting errors may be introduced. Fitting methods such as K-mean algorithms [12] reduce the fitting errors, but still do not guarantee optimality. Generalized posynomial form [2] does not solve this issue.

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 and smoothness required to ensure quick global convergence. As a result, it is necessary to modify the data such that both convexity and smoothness properties can be guaranteed. In fairly recent literature numerically convex tables have been proposed [15]. These tables are created optimally by minimizing the perturbation of data to make them numerically convex. But since these tables are numerical, it is extremely important to make the table data smooth, and yet preserve its convexity. Smoothness will ensure that the convex optimizer behaves in a predictable way and converges quickly to the global optimal point. The smoothing technique proposed in [15] cannot guarantee continuous differentiability, nor can it preserve the convexity of the data.

In this paper, we propose to simultaneously create optimal numerically convex look-up tables and guarantee smoothness in the data, without explicit analytical form. Smoothness in the table data will enable the convex optimizer to converge predictably. We show that numerically "convexifying" and "smoothing" 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 and smoothness is still ensured. As a result, convex optimization algorithms can be applied.

We present our convexifying and smoothing results on industrial cell libraries. *ConvexSmooth* shows 14X reduction in fitting error over a well-developed posynomial fitting algorithm. Note 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 organization of the paper is as follows. In Section 2 we provide some general background on convexity, smoothness and semidefinite programming. We briefly survey the *ConvexFit* formulation and smoothness technique from [15] in sections 3.1 and 3.2. We also discuss the disadvantages of this technique. We propose our *ConvexSmooth* problem formulation in 3.3. In section 4 we provide experimental results of *ConvexSmooth* 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 concepts of convexity, smoothness and semidefinite programming is introduced.

2.1 Convexity, Hessian, and Smoothness

We now introduce the definition of a convex function. 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. In a convex function, every local minimizer is also a global minimizer.

A smooth function is one that is infinitely differentiable, or has derivatives of all finite orders. A function is called C^1 if it has a derivative that is a continuous function. Such functions are also called continuously differentiable. For quick and guaranteed convergence to a global optimal point, a convex function must be at least continuously differentiable. Smoothness ensures that the function is free from kinks and jumps and helps algorithms to make good choice for search directions.

2.2 Semidefinite Programming

Now we introduce semidefinite programming which will be used to solve our optimization problem. 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 :

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

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

3. CONVEX FITTING PROBLEM FORMULATION

In this section, we briefly discuss the minimum-error convex fitting problem and smoothing technique proposed in [15].

3.1 Convex Fitting Problem Formulation

We now illustrate the minimum-error convex fitting problem formulation. We then propose our problem formulation for $ConvexSmooth$ in subsection 3.3

Given an analytical or numerical function $g(\mathbf{x})$, let $f(\mathbf{x}) - g(\mathbf{x}) = \delta'(\mathbf{x})$, then $\delta'(\mathbf{x})$ is the perturbation of $g(\mathbf{x})$, the task of $ConvexFit$ is to minimize the perturbation of $g(\mathbf{x})$ to make the hessian of $f(\mathbf{x})$ positive semidefinite. The minimum-error convex fitting problem is defined as follows:

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

Since $|\ \delta(\mathbf{x})\ |$ is not a linear function of $\mathbf{x}$, $-\delta(\mathbf{x}) \leq \delta'(\mathbf{x}) \leq \delta(\mathbf{x})$ is used, where $\delta(\mathbf{x}) \geq 0$. Under this transformation, the following formulation is obtained:

$$\begin{aligned} & ConvexFit': \\ & \qquad minimize \quad \sum_{\mathbf{x} \in DOMg} \delta(\mathbf{x}) \\ & \qquad subject\ to \quad \nabla^2(g(\mathbf{x}) + \delta'(\mathbf{x})) \succeq 0, \\ & \qquad\qquad\qquad -\delta(\mathbf{x}) \leq \delta'(\mathbf{x}) \leq \delta(\mathbf{x}), \\ & \qquad\qquad\qquad \delta(\mathbf{x}) \geq 0, \\ & \qquad\qquad\qquad \mathbf{x} \in DOMg \end{aligned} \tag{4}$$

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 of ConvexFit

Here we briefly discuss the smoothing of $ConvexFit$ and its disadvantages. We then propose a new problem formulation in 3.3 to solve the inherent problems of this technique.

Let f be a discrete function of $\mathbf{x} = [x_1, x_2, ..., x_n]^T$. The following quadratic form is used 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.

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. The 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 $(x_{1l}, x_{2l}, ...)$,

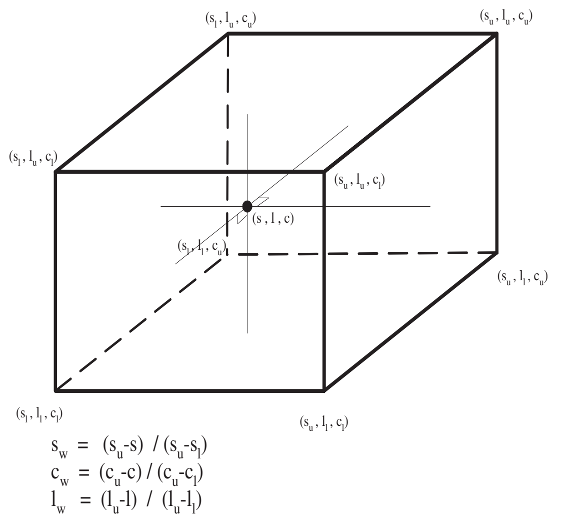

Figure 1: ConvexFit Smoothing for 3-dimension

$(x_{1u}, x_{2l}, ...), (x_{1l}, x_{2u}, ...), (x_{1u}, x_{2u}, ...)....$ where $x_{1l} < x_1 < x_{1u}$, $x_{2l} < x_2 < x_{2u}, ...$ and so on.

Weight $x_{1w} = (x_{1u} - x_1)/(x_{1u} - x_{1l})$. Similarly, weight $x_{2w}, x_{3w}, ...x_{nw}$. The product $(x_{1w} \times x_{2w} \times x_{3w} ... \times x_{nw})$ is used as the weight of point $(x_{1l}, x_{2l}, x_{3l}, ..., x_{nl})$. Similarly $((1 - x_{1w}) \times x_{2w} \times x_{3w} ... \times\ x_{nw})$ is used as weight of point $(x_{1u}, x_{2l}, x_{3l}, ..., x_{nl})$ and so on. The sum $\sum(value \times weight)$ is used as the smoothed value of point $(x_1, x_2, .., x_n)$. Figure 1 illustrates the weight calculation for 3-dimensional space.

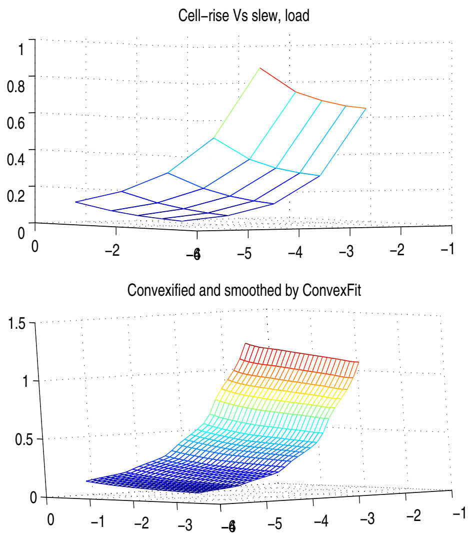

Figure 2: Original data and Smoothing by ConvexFit for cell INV

This smoothing algorithm has many disadvantages. Firstly it adds additional data points by interpolation to an existing numerically convex data (from 3.1). There is no guarantee that the new points will preserve the convexity of the original data, as no such constraint is imposed on the new data points. Also the interpolated data points may not make the table sufficiently smooth, or in other words the continuous differentiability cannot be guaranteed from this smoothing technique. Fig. 2 illustrates the model for cell INV which is convexified and then smoothed by the above technique. It can be seen that smoothing does not preserve the convexity of the convexified model. We now propose an algorithm by which the numerical model will converge to a continuously differentiable convex model.

3.3 Converging to a continuously differentiable model by ConvexSmooth

In a convex optimization problem, the optimizer needs a smooth continuously differentiable convex model to gradually reach the global optimal point. At first we introduce a term NSI or non-smoothness index. NSI for a discrete function $f(\mathbf{x})$ is defined as

$$\begin{aligned} NSI(f) = \max\ \mid \nabla_i f(\mathbf{x}) - \nabla_i f(\mathbf{y}) \mid&, \\ i \in [1, 2, ..., n]&, \\ \mathbf{x}, \mathbf{y} \in adj\ pnts\ in\ DOM f& \end{aligned}$$

The smaller the value of NSI, the higher is the smoothness in a curve. ϵ the maximum allowable non-smoothness will be an input to our algorithm. We now formulate our optimization problem to ensure continuous differentiability in addition to convexity.

- **Strategic addition of points**

  Let g be a discrete function of $\mathbf{x} = [x_1, x_2, ..., x_n]^T$. We check all pairs of adjacent data points. When $\mid \nabla_i g(\mathbf{x}) - \nabla_i g(\mathbf{y}) \mid > \epsilon$ for adjacent points $\mathbf{x}, \mathbf{y}$ for a given $\epsilon > 0$, we introduce additional points between $\mathbf{x}, \mathbf{y}$. We use quadratic interpolation to generate data values of g at intermediate points $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$.

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

  where $C = g(\mathbf{x})$, $\mathbf{b} = [\frac{\partial g(\mathbf{x})}{\partial x_1}, \frac{\partial g(\mathbf{x})}{\partial x_2}, ..., \frac{\partial g(\mathbf{x})}{\partial x_n}]^T$ and H_g is the Hessian of g.

- **Perturb data to make it continuously differentiable**

  Let $g'(\mathbf{x}')$ be the new function with the additional points generated from the above step. We now introduce perturbation in $g'(\mathbf{x}')$ to make it continuously differentiable and also make the hessian of the perturbed function positive semidefinite. Let $f'(\mathbf{x}') - g'(\mathbf{x}') = \phi'(\mathbf{x}')$, $\phi'(\mathbf{x}')$ being the perturbation. The problem is formulated below:

$$\begin{aligned} &ConvexSmooth: \\ &\quad minimize \qquad \textstyle\sum_{\mathbf{x} \in DOM g} \mid \phi'(\mathbf{x}) \mid \\ &\quad subject\ to \qquad \nabla^2(g'(\mathbf{x}') + \phi'(\mathbf{x}')) \succeq 0, \\ &\qquad\qquad -\epsilon < \nabla_i f'(\mathbf{x}') - \nabla_i f'(\mathbf{y}') < \epsilon, \\ &\qquad\qquad i \in [1, 2, ..., n], \\ &\qquad\qquad \forall\, \mathbf{x}', \mathbf{y}' \in adj\ pnts\ in\ DOM f', \\ &\quad for \qquad given\ \epsilon > 0, \end{aligned} \tag{5}$$

Again the nonlinear function $\mid \phi'(\mathbf{x}) \mid$ can be transformed by the following formulation

$$
\begin{array}{lll}
\multicolumn{3}{l}{ConvexSmooth' :} \\
\quad minimize & \sum_{\mathbf{x}\in DOM_g} \phi(\mathbf{x}) & \\
\quad subject\ to & \nabla^2(g'(\mathbf{x}') + \phi'(\mathbf{x}')) \succeq 0, & \\
 & -\phi(\mathbf{x}') \leq \phi'(\mathbf{x}') \leq \phi(\mathbf{x}'), & \\
 & -\epsilon < \nabla_i f'(\mathbf{x}') - \nabla_i f'(\mathbf{y}') < \epsilon, & \\
 & i \in [1, 2, ..., n], & \\
 & \forall\, \mathbf{x}', \mathbf{y}' \in adj\ pnts\ in\ DOMf', & \\
\quad for & given\ \epsilon > 0, & (6)
\end{array}
$$

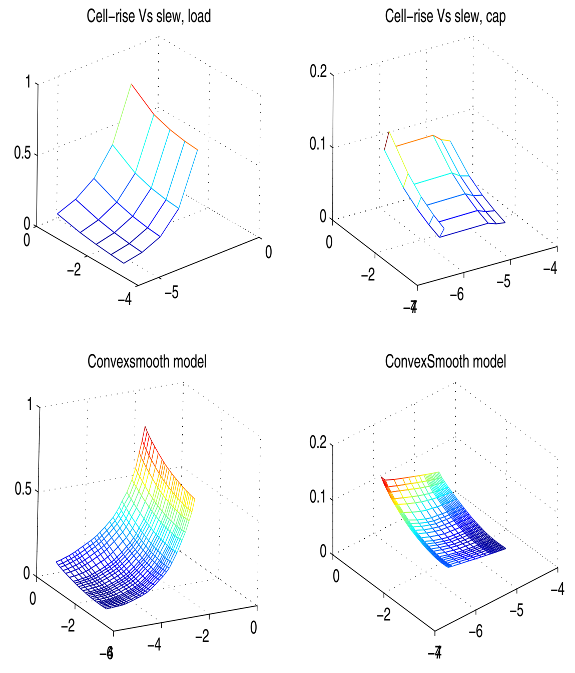

Figure 3: Plot of original data and ConvexSmooth model for cell INV

Hence now we get a numerically convex data model which is also sufficiently smooth. Figure 3 illustrates the model developed from our algorithm. The $ConvexSmooth'$ formulation can be easily fitted into the dual form (D) of semidefinite programming framework [3]. Since semidefinite programming is a convex optimization problem, the $ConvexSmooth'$ problem can be optimally solved by any semidefinite programming solver. The topic of shape preserving interpolation and smoothing of a numerical model is an open area of research. We plan to explore some more intricate and accurate techniques in this area in our future work.

3.4 Posynomial Fitting Procedure for comparison

We compare $ConvexSmooth$ with the posynomial modelling technique. We briefly describe posynomial modelling and our approach here.

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:

$$
\begin{array}{lll}
PosynomialFit: & minimize & \sum_{m=1}^{z}((\sum_{j=1}^{k_m} c_j \prod_{i=1}^{n} x_{mi}{}^{\alpha_{ij}}) - b_m)^2 \\
 & subject\ to & c_j \geq 0 \qquad\qquad (7)
\end{array}
$$

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. 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.

Approach - 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:

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

Table 1: Cell-Rise Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.883	0.035	0.232	0.011
AO22	8.892	0.117	0.875	0.026
AOI222	9.214	0.137	0.187	0.010
INV	9.427	0.087	0.052	0.004
MAOI1	8.100	0.125	0.403	0.018
MUX4	5.900	0.108	0.032	0.005
NR4	3.213	0.088	0.097	0.007
OA12	3.523	0.084	0.348	0.013
OR3B2	4.009	0.087	0.578	0.016
XOR2	2.537	0.064	1.504	0.025

4. EXPERIMENTAL RESULTS ON INDUSTRIAL CELL LIBRARY

In this section, we first introduce how to apply $ConvexSmooth$ to a popular industrial cell library to generate convex delay and transition time models. Next, we present the experimental comparison between $PosynomialFit$ and $ConvexSmooth$

algorithms. Then we discuss the tradeoff between smoothness and execution time.

4.1 Applying ConvexSmooth in generating standard cell delay models

Our *ConvexSmooth* formulation (6) 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 and smooth. The perturbed function obtained by running *ConvexSmooth* is our generated delay model for the standard cells.

Table 2: Cell-Fall Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.130	0.013	0.049	0.004
AO22	1.161	0.043	0.070	0.008
AOI222	1.689	0.064	0.067	0.008
INV	1.768	0.039	0.022	0.003
MAOI1	1.017	0.048	0.469	0.024
MUX4	0.926	0.046	0.067	0.006
NR4	0.681	0.052	0.017	0.004
OA12	0.506	0.029	0.049	0.003
OR3B2	1.274	0.053	0.043	0.006
XOR2	0.395	0.029	0.078	0.004

Table 3: Rise-Transition Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	4.818	0.083	2.845	0.043
AO22	44.886	0.263	5.46	0.070
AOI222	52.601	0.352	2.01	0.041
INV	42.715	0.184	0.546	0.016
MAOI1	39.820	0.286	2.06	0.055
MUX4	33.874	0.279	0.200	0.012
NR4	17.178	0.219	1.013	0.027
OA12	18.054	0.169	2.618	0.048
OR3B2	22.853	0.216	3.499	0.049
XOR2	16.216	0.169	0.863	0.015

4.2 Comparison of PosynomialFit and ConvexSmooth

We now present the experimental results of *ConvexSmooth* 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 *ConvexSmooth*. *PosynomialFit* was implemented in C++ using the CFSQP solver [4]. All the procedures were performed after logarithmically transforming the data points in the lookup table. The number of monomial terms k_m in *PosynomialFit* was fixed to 5. All experiments were performed on a PC with 1.40GHz Microprocessor, 1.00 GB RAM and 40 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 1, 2, 3, 4 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 calculated 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 *ConvexSmooth* shows more than 14X reduction in fitting error over *PosynomialFit* and yet it can guarantee convexity and sufficient smoothness.

Table 4: Fall-Transition Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.854	0.042	0.141	0.008
AO22	5.531	0.089	0.565	0.019
AOI222	9.531	0.158	0.058	0.006
INV	5.317	0.066	0.036	0.003
MAOI1	5.218	0.108	0.804	0.029
MUX4	4.306	0.093	0.026	0.004
NR4	2.198	0.075	0.056	0.006
OA12	2.237	0.058	0.203	0.009
OR3B2	7.531	0.124	0.704	0.028
XOR2	2.943	0.069	0.863	0.015

Table 5: Execution times

Mode	ConvexFit	ConvexSmooth			
	$167pts$ (sec)	$167pts$ (sec)	$294pts$ (sec)	$546pts$ (sec)	$1014pts$ (sec)
Cell-Rise	2	16	46	87	333
Cell-Fall	2	17	48	90	332
Rise-Trans	3	17	41	85	322
Fall-Trans	2	17	45	99	339

4.3 Tradeoff between smoothness and execution time

Table 5 presents the runtimes of *ConvexSmooth* for different number of data points per cell. The execution times are the average execution time per cell in seconds. The column headings show the average number of data points per cell. The *ConvexFit* column shows the execution times for *ConvexFit* with the original number of data points. Note that for the same number of data points, *ConvexSmooth* requires higher execution time, as we guarantee smoothness by additional constraints, as per equation 6. The higher the number of data points inserted in 3.3 the smaller is the NSI, and the greater is the smoothness achieved in our model. But higher data points also increases the execution time of our algorithm. Figure 4 shows a plot of execution time against the number of data points for cell INV. It can be seen that the higher the data points, the higher is the execution time. Often depending on the size, complexity and type of our optimization problem, we may not need more than 2X/4X additional point addition. Hence the highest permissible value of ϵ should be found out for a specific type of problem to minimize the execution time of modelling.

5. CONCLUSION

Convex optimization has gained popularity due to its capability to reach global optimum in a reasonable amount of time. Table data is often fitted into analytical forms like

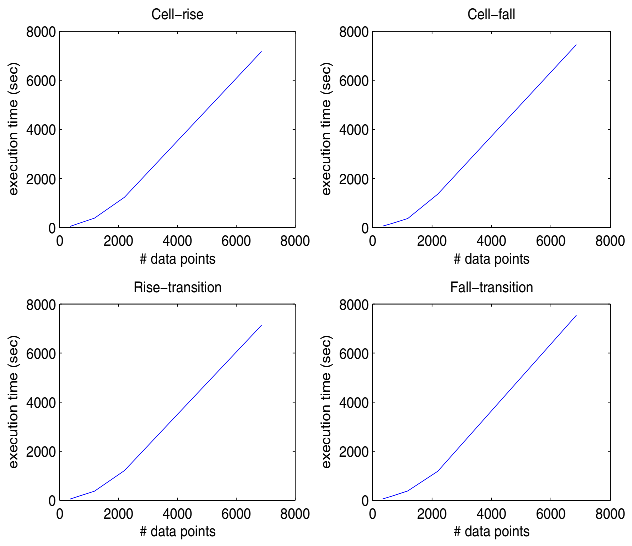

Figure 4: Plot of execution time Vs data points for cell INV

posynomials to make them convex. However, fitting the look-up tables into posynomial forms with minimum error itself may not be a convex optimization problem and hence excessive fitting errors may be introduced. In recent literature numerically convex tables have been proposed. These tables are created optimally by minimizing the perturbation of data to make them numerically convex. But since these tables are numerical, it is extremely important to make the table data smooth, and yet preserve its convexity. In this paper, we propose to simultaneously create optimal numerically convex look-up tables and at the same time guarantee smoothness in the data, without explicit analytical form. We show that numerically "convexifying" and "smoothing" the table data with minimum perturbation can be formulated as a convex semidefinite optimization problem and hence optimality can be reached in polynomial time. We present our convexifying and smoothing results on industrial cell libraries. We demonstrate 14X reduction in fitting error over a well-developed posynomial fitting algorithm.

6. 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] 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.

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

[15] Sanghamitra Roy, Weijen Chen and Charlie Chung-Ping Chen,"ConvexFit: An Optimal Minimum-Error Convex Fitting and Smoothing Algorithm with Application to Gate Sizing", in *Proceedings of the International Conference on Computer Aided Design*, 2005.