SmartSmooth: A linear time convexity preserving smoothing algorithm for numerically convex data with application to VLSI design

Sanghamitra Roy
ECE Department
University of Wisconsin-Madison
roy1.wisc.edu

Charlie Chung-Ping Chen
EE Department
National Taiwan University
cchen@cc.ee.ntu.edu.tw

ABSTRACT

Convex optimization problems are very popular in the VLSI design society due to their guaranteed convergence to a global optimal point. While optimizing tabular data, significant fitting efforts are required to fit the data into convex form. Fitting the tables into analytically convex forms like posynomials, suffers from excessive fitting errors, as the fitting problem may be non-convex. In recent literature optimal numerically convex tables have been proposed. Since these tables are numerical, it is extremely important to make the table data smooth, and yet preserve its convexity. The smoothness ensures that the convex optimizer behaves predictably and converges quickly to the global optimal point. The existing smoothing techniques either cannot preserve convexity, or require very high execution time. In this paper, we propose a linear time algorithm to smoothen a given numerically convex data and at the same time preserve convexity. Our proposed algorithm SmartSmooth can smoothen the data in linear time without introducing any additional error on the numerically convex data. We present our SmartSmooth results on industrial cell libraries. SmartSmooth when applied on convex tables produced by ConvexFit shows a 30X reduction in fitting square error over a 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 fit look-up table data into convex form. Fitting the tables into analytically explicit convex functional form such as posynomials [2, 5], suffers from high fitting errors as the fitting problem may be non-convex. Fitting methods such as K-mean algorithms [12] reduce the fitting errors, but still do not guarantee optimality. It is important to have an optimal solution to the fitting problem.

Another orthogonal approach is to directly use the look-up table data for optimization. Using finite difference method, we can still obtain sensitivity and even hessian which are sufficient for optimization usage. However, it is critical to modify the data such that both convexity and smoothness properties can be guaranteed to ensure fast global convergence. In fairly recent literature, *ConvexFit* an algorithm to generate numerically convex tables has 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. A simultaneous convex fitting and smoothing algorithm *ConvexSmooth* was proposed by Roy et al.[16] but it tried to solve a large semidefinite optimization problem with higher data points resulting in high execution time.

In this paper, we propose an algorithm to smoothen a given numerically convex data in linear time, while preserving the qualitative shape properties of the original data, in other words convexity. After a lookup table is fitted into a numerically convex model, our proposed algorithm *SmartSmooth* can smoothen the data in linear time without introducing any additional error on the numerically convex data. Hence *SmartSmooth* can also be described as a convexity preserving interpolation technique. We formulate the convexity preserving smoothing problem as a series of inexpensive local semidefinite optimizations which can be solved in linear time.

We present our *SmartSmooth* results on industrial cell libraries. *ConvexFit* followed by *SmartSmooth* can generate numerically convex and smooth data while giving 30X reduction in fitting square error over a well-developed posynomial fitting algorithm, and 3X reduction in fitting square error over *ConvexSmooth*.

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 section 3. We propose our algorithm *SmartSmooth* problem formulation in section 4. In section 5 we briefly describe *PosynomialFit*, and *ConvexSmooth*, previous techniques with which we compare our algorithm. In section 6 we provide experimental results of *ConvexFit* + *SmartSmooth* on industrial cell libraries. We conclude our discussion in section 7.

2. FUNDAMENTAL CONCEPTS

In this section, the fundamental concepts of convexity, smoothness and semidefinite programming is introduced.

2.1 Convexity, Hessian, and Smoothness

1-4244-0630-7/07/$20.00 ©2007 IEEE.

We now introduce the definition of a convex function. A function $f(x)$ is convex if

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

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.

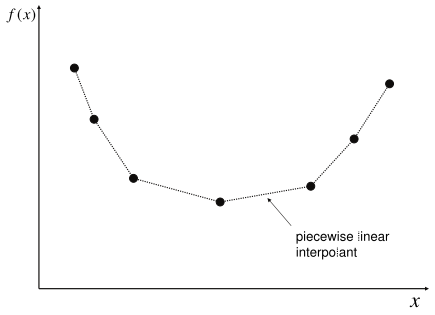

Figure 1: A numerically convex function in 1-dimension

A data set is numerically convex if and only if there exists a convex piecewise linear interpolant to the data. Figure 1 shows an example of a numerically convex function in one dimension. Note that every analytically convex function is also numerically convex but the converse is not true.

A function is called C^1 if it has a derivative that is a continuous function. A function f is smooth at x if given $\epsilon > 0$ there is a $\delta > 0$ such that

$$|\, y-x \,| < \delta \Rightarrow |\, \nabla f(y) - \nabla f(x) \,| < \epsilon$$

For quick and guaranteed convergence to a global optimal point, a convex function must be at least continuously differentiable.

2.2 Semidefinite Programming

Now we introduce the dual form of semidefinite programming which we will use to solve our optimization problem. The dual form (D) of SDP is given by :

$$\begin{aligned} (D)\quad & maximize & & \sum_{i=1}^{m} b_i y_i \\ & subject\ to & & C_j - \sum_{i=1}^{m} A_{i,j} y_i \succeq 0, \\ & & & j = 1, ..., n_b \end{aligned} \qquad (1)$$

where b_i,y_i are scalars, $A_{i,j}$ and C_j are symmetric matrices of the same dimension.

2.3 Adjacent points in a multidimensional grid structure

We define adjacent points ($adj\ pnts_i$) for an n-dimensional regular or irregular grid structure. Adjacent points are defined with respect to a particular direction i. Two points are $adj\ pnts_i$ in the i_{th} direction if they have identical coordinates for all the other $(n-1)$ directions and they have adjacent coordinates in the i_{th} direction. Thus if f is a numerical function of $\mathbf{x} = [x_1, x_2, ..., x_n]^T$, then the points $(x_1(k), ..., x_i(j), ..., x_n(l))$ and $(x_1(k), ..., x_i(j+1), ..., x_n(l))$ are two $adj\ pnts_i$.

3. CONVEXFIT

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

3.1 Convex Fitting

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:

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

Since the domain 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 $g(\mathbf{x})$.

3.2 Smoothing of ConvexFit

Let f be a numerical 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 not on the boundary of the hypercubes, has 2^n surrounding points. The smoothing algorithm calculates the value at an intermediate point by using a weighted sum of the values obtained from each surrounding point.

4. SMARTSMOOTH: LINEAR TIME CONVEXITY PRESERVING SMOOTHING OF CONVEXFIT

In this section, we discuss the disadvantages of the previous smoothing techniques and propose $SmartSmooth$ as a succeeding step to $ConvexFit$.

The smoothing algorithm mentioned in subsection 3.2 has many disadvantages. Firstly, it adds additional data points by interpolation to an existing numerically convex data. There is no guarantee that the new points will preserve the

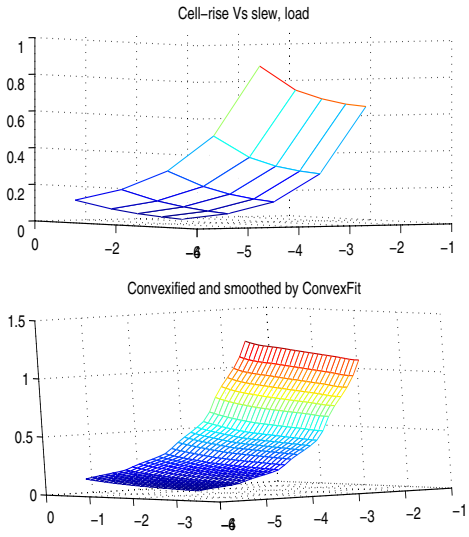

Figure 2: Original data and Smoothing by ConvexFit for cell INV

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, 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 model. A simultaneous convex fitting and smoothing algorithm *ConvexSmooth* was recently proposed [16]. This method had to solve increasingly large semidefinite optimization problems for higher number of data point insertion. Hence it was expensive in terms of execution time. We now propose *SmartSmooth* a linear time convexity preserving smoothing algorithm.

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

$$NSI(f) = \max \mid \nabla_i f(\mathbf{x}) - \nabla_i f(\mathbf{y}) \mid, \quad \mathbf{x}, \mathbf{y} \in adj\ pnts_i\ in\ DOM f \quad i \in [1, 2, ..., n],$$

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

- **Addition of points by quadratic interpolation**

  Let g be a numerically convex function of $\mathbf{x} = [x_1, .., x_n]^T$ We use quadratic interpolation to generate additional 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.

- **Locally perturb newly added data to make it continuously differentiable and preserve convexity**

  Let $g(\mathbf{x}')$ be the new function with the additional points generated from the above step. We now divide the domain of $g(\mathbf{x}')$ into smaller hypercubes. Let $g_{h1}(\mathbf{x'_{h1}})$, $g_{h2}(\mathbf{x'_{h2}})$, ... , $g_{hk}(\mathbf{x'_{hk}})$ be the corresponding subfunctions in the hypercubes $\mathbf{x'_{h1}}$, $\mathbf{x'_{h2}}$, ..., $\mathbf{x'_{hk}}$. Instead of running an optimization problem on the entire data set $g(\mathbf{x}')$, we solve an optimization problem for each subfunction. As our original data points is numerically convex, this problem becomes feasible. Hence we can make the new function smooth and drastically reduce execution time of smoothing. For each hypercube we introduce perturbation in only the newly added points to make the perturbed subfunction continuously differentiable and also make the hessian positive semidefinite. Let $f_{hi}(\mathbf{x'_{hi}}) = g_{hi}(\mathbf{x'_{hi}}) + \phi'_{hi}(\mathbf{x'_{hi}})$, $\phi'(\mathbf{x'_{hi}})$ being the perturbation. Note that $\phi'(\mathbf{x_{hi}}) = 0$ for $x_{hi} \subset x'_{hi}$, where x_{hi} consists of the original data points in $DOMg_{hi}$. In other words, the original data points are given zero perturbation. The problem is formulated below for each hypercube x'_{hi}:

$$\begin{aligned} &SmartSmooth: \\ &\quad minimize && \textstyle\sum_{\mathbf{x}\in DOMg_{hi}} \mid \phi'_{hi}(\mathbf{x}) \mid \\ &\quad subject\ to && \nabla^2(g_{hi}(\mathbf{x'_{hi}}) + \phi'_{hi}(\mathbf{x'_{hi}})) \succeq 0, \\ &&& -\epsilon < \nabla_j f_{hi}(\mathbf{x'_{hi}}) - \nabla_j f_{hi}(\mathbf{y'_{hi}}) < \epsilon, \\ &&& \forall\, \mathbf{x'_{hi}}, \mathbf{y'_{hi}} \in adj\ pnts_j\ in\ DOM f_{hi}, \\ &&& j \in [1, 2, ..., n], \\ &\quad for && given\ \epsilon > 0, \end{aligned} \tag{3}$$

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

$$\begin{aligned} &SmartSmooth': \\ &\quad minimize && \textstyle\sum_{\mathbf{x}\in DOMg_{hi}} \phi_{hi}(\mathbf{x}) \\ &\quad subject\ to && \nabla^2(g_{hi}(\mathbf{x'_{hi}}) + \phi'_{hi}(\mathbf{x'_{hi}})) \succeq 0, \\ &&& -\phi_{hi}(\mathbf{x'_{hi}}) \le \phi'_{hi}(\mathbf{x'_{hi}}) \le \phi_{hi}(\mathbf{x'_{hi}}), \\ &&& \phi_{hi}(\mathbf{x'_{hi}}) \ge 0, \\ &&& -\epsilon < \nabla_j f_{hi}(\mathbf{x'_{hi}}) - \nabla_j f_{hi}(\mathbf{y'_{hi}}) < \epsilon, \\ &&& \forall\, \mathbf{x'_{hi}}, \mathbf{y'_{hi}} \in adj\ pnts_j\ in\ DOM f_{hi}, \\ &&& j \in [1, 2, ..., n], \\ &\quad for && given\ \epsilon > 0, \end{aligned} \tag{4}$$

The first constraint in $SmartSmooth'$ ensures that the new function remains convex by enforcing a positive semidefinite hessian. The fourth constraint ensures that the non-smoothness in the new function is less than the maximum allowable NSI parameter ϵ. Hence, we get a numerically convex data model which is also sufficiently smooth. Note that since the number of data points in each local semidefinite optimization is small (constant), our overall execution time is linear with respect to the number of data points. The $SmartSmooth'$ formulation can be easily fitted into the dual form (D) of semidefinite programming framework [3] and

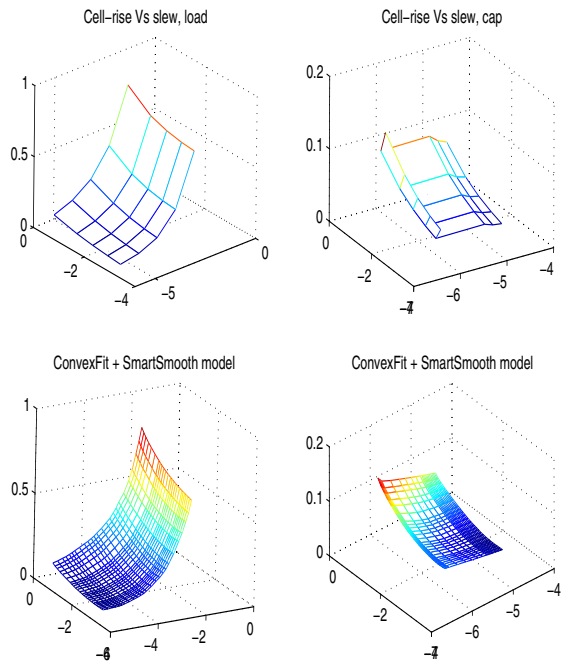

Figure 3: Plot of original data and ConvexFit + SmartSmooth model for cell INV

can be optimally solved by any semidefinite programming solver. Figure 3 illustrates the model developed from our algorithm.

5. POSYNOMIAL FITTING AND CONVEXSMOOTH FOR COMPARISON

We compare *ConvexFit + SmartSmooth* with the two algorithms *PosynomialFit* and *ConvexSmooth*. We briefly describe both the techniques here.

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

$$PosynomialFit: \quad minimize \quad \sum_{m=1}^{z}((\sum_{j=1}^{k_m} c_j \prod_{i=1}^{n} {x_{mi}}^{\alpha_{ij}}) - b_m)^2$$
$$subject\ to \quad c_j \geq 0 \qquad (5)$$

where z is the number of sets of tunable parameters , n is the number of tunable parameters, $x_{mi} \in \Re$ 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.

ConvexSmooth is a simultaneous convex fitting and smoothing algorithm. In this algorithm additional data points are initially added to the lookup table by quadratic interpolation. Then the new table is convexified and smoothed by running a global semidefinite optimization problem. This method has to solve increasingly large semidefinite optimization problems for higher number of data point insertion. Hence it is expensive in terms of execution time.

Table 1: Cell-Rise Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth		CFit+SSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.883	0.035	0.232	0.011	0.287	0.016
AOI222	9.214	0.137	0.187	0.010	0.144	0.008
BUF	3.861	0.047	11.273	0.081	1.412	0.026
INV	9.427	0.087	0.052	0.004	0.052	0.004
MAOI1	8.100	0.125	0.403	0.018	0.024	0.004
MUX4	5.900	0.108	0.032	0.005	0.032	0.004
ND3	1.774	0.068	1.111	0.029	0.168	0.014
OA12	3.523	0.084	0.348	0.013	0.336	0.016
OR3B2	4.009	0.087	0.578	0.016	0.074	0.004
XOR2	2.537	0.064	1.504	0.025	0.000	0.000

6. EXPERIMENTAL RESULTS ON INDUSTRIAL CELL LIBRARY

We now present the experimental results of *ConvexFit + SmartSmooth*, *PosynomialFit* and *ConvexSmooth* 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 perform our experiments using 67 combinational cells from this library. We use the DSDP5.7 [3] solver in C to run our semidefinite optimizations *ConvexFit*, *SmartSmooth* and *ConvexSmooth*. *PosynomialFit* is implemented in C++ using the CFSQP solver [4]. All the procedures are performed after logarithmically transforming the data points in the lookup table. The number of monomial terms k_m in *PosynomialFit* is fixed to 5. All experiments are performed on a PC with 1.40GHz Pentium IV Microprocessor, 1.00 GB RAM and 40 GB hard drive running Windows XP. Experiments are performed for cell-rise, cell-fall, rise-transition and fall-transition look-up tables for each cell, and for one input pin per cell. These look-up tables have a size ranging from 49 to 392 data points.

Table 2: Cell-Fall Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth		CFit+SSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.130	0.0130	0.049	0.004	0.043	0.005
AOI222	1.689	0.0637	0.067	0.008	0.068	0.007
BUF	0.524	0.0189	3.99	0.054	0.052	0.003
INV	1.768	0.0391	0.022	0.003	0.035	0.005
MAOI1	1.017	0.0483	0.469	0.024	0.078	0.008
MUX4	0.926	0.0462	0.067	0.006	0.066	0.006
ND3	1.506	0.0672	0.241	0.017	0.048	0.008
OA12	0.506	0.0289	0.049	0.003	0.051	0.004
OR3B2	1.274	0.0530	0.043	0.006	0.003	0.001
XOR2	0.395	0.0292	0.078	0.004	0.000	0.000

6.1 Comparison of PosynomialFit, ConvexSmooth and ConvexFit + SmartSmooth

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 space constraint). 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

Table 3: Rise-Transition Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth		CFit+SSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	4.818	0.0832	2.845	0.043	1.213	0.029
AOI222	52.601	0.3521	2.01	0.041	0.701	0.021
BUF	18.392	0.0990	7.586	0.021	7.410	0.059
INV	42.715	0.1843	0.546	0.016	0.206	0.010
MAOI1	39.820	0.2859	2.06	0.055	1.14	0.029
MUX4	33.874	0.2786	0.200	0.012	0.182	0.010
ND3	9.561	0.1526	5.56	0.059	0.186	0.008
OA12	18.054	0.1689	2.618	0.048	1.88	0.042
OR3B2	22.853	0.2164	3.499	0.049	0.649	0.022
XOR2	16.216	0.1695	0.169	0.015	0.002	0.001

points. It can be observed that *ConvexFit + SmartSmooth* shows more than 30X reduction in fitting square error over *PosynomialFit*, and 3X reduction in fitting square error over *ConvexSmooth*. This is because *SmartSmooth* adds new points to the model generated by *ConvexFit* and only perturbs the newly added points. Hence, it does not introduce additional error on the data points generated by *ConvexFit*. (Note that all the errors are calculated with respect to the original points in the look-up table.)

Table 4: Fall-Transition Fitting Errors Comparison

Cell name	PosynomialFit		ConvexSmooth		CFit+SSmooth	
	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)	SE $(ns)^2$	AE (ns)
AN2	0.854	0.0421	0.141	0.008	0.136	0.009
AOI222	9.531	0.1582	0.058	0.006	0.051	0.006
BUF	2.048	0.0355	4.35	0.046	0.706	0.017
INV	5.317	0.0664	0.036	0.003	0.034	0.003
MAOI1	5.218	0.1081	0.804	0.029	0.216	0.012
MUX4	4.306	0.0933	0.026	0.004	0.026	0.004
ND3	6.414	0.1329	0.237	0.013	0.000	0.000
OA12	2.237	0.0576	0.203	0.009	0.200	0.012
OR3B2	7.531	0.1242	0.704	0.028	0.411	0.018
XOR2	2.943	0.0690	0.863	0.015	0.000	0.000

Table 5: Execution times in seconds

Mode	data points							
	196	196	1183	3610	8125	1183	3610	8125
	PFit	CFit	ConvexSmooth			CFit+SSmooth		
CR	17s	2s	116s	489s	3024s	5s	11s	46s
CF	19s	1s	117s	560s	2470s	4s	10s	44s
RT	18s	2s	112s	403s	2551s	5s	10s	44s
FT	18s	2s	117s	461s	3237s	5s	11s	46s

6.2 Tradeoff between smoothness and execution time

Table 5 shows the runtimes of algorithms *PosynomialFit*, *ConvexSmooth* and *ConvexFit + SmartSmooth* 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 *PFit* and *CFit* columns show the execution times for *PosynomialFit* and *ConvexFit* with the original number of data points in the look-up table. The higher the number of data points inserted in section 4 the smaller is the *NSI*, and the greater is the smoothness achieved in our model. But

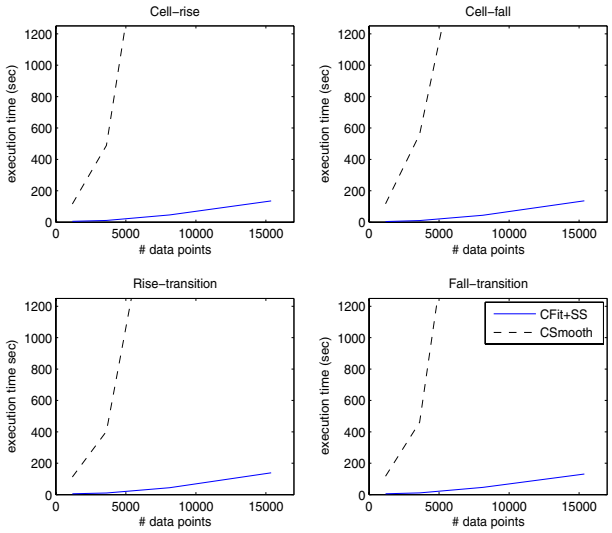

Figure 4: Plot of execution time Vs data points for ConvexSmooth and ConvexFit + SmartSmooth

higher data points also increases the execution time of our algorithm. It can be seen that *ConvexFit + SmartSmooth* with up to 20X point addition still requires a smaller execution time than *PosynomialFit*. With 6X point addition we demonstrate a 4X speedup over *PosynomialFit*. We also demonstrate a 56X speedup over *ConvexSmooth* which clearly shows we have been able to tackle the higher execution time problem in providing higher smoothness. Often depending on the size, complexity and type of our optimization problem, we may not need more than 4X/10X additional point insertion in the look-up table. Hence the highest permissible value of ϵ should be found out for a specific type of problem, to minimize the execution time of modeling. Figure 4 shows the plot of execution time against the average number of data points per cell for *ConvexSmooth* and *ConvexFit + SmartSmooth*. *ConvexSmooth* requires very high execution time while *ConvexFit + SmartSmooth* requires linear execution time with respect to the number of data points.

6.3 Benefit of smoothing in faster convergence of the gate sizing optimization

We test the performance of our smoothing algorithm in the gate sizing optimization problem. We perform experiments on ISCAS85 benchmark circuits using our gate sizing tool implemented in C++. These circuits have 214 to 3512 gates. We perform gate sizing using two different types of numerically convex models for the gate delay. The first is the numerically convex table generated by *ConvexFit* and we use linear interpolation to calculate intermediate points. Linear interpolation always preserves convexity , but it may not make the table sufficiently smooth, as it creates non-smooth corners. The second table used for the gate sizing optimization is the table generated using *ConvexFit* and smoothened by *SmartSmooth*. Table 6 shows the convergence time of optimization using the two lookup tables. It can be seen that smoothing by *SmartSmooth* can speed up

the gate sizing optimization by 1.76X on average, for these benchmarks. The quality (circuit delay) of the optimized designs have been found to be comparable using both the two models. Thus we have shown that smoothing helps in faster convergence of the convex optimizer.

Table 6: Convergence time for sizing various circuits

circuit	Linear (sec)	SmartSmooth (sec)
C432	11	4
C499	3	1
C880	8	15
C1355	14	9
C1908	5	3
C2670	53	26
C3540	66	64
C5315	55	39
C6288	1952	347
C7552	169	110

7. CONCLUSION

Convex optimization problems are very popular in the VLSI design society due to their capability to reach global optimum in a reasonable amount of time. Table data needs to be fitted into convex forms to be used in convex optimization problems. Fitting the tables into analytically convex forms like posynomials, suffers from high fitting errors as the fitting problem may be non-convex. In recent literature numerically convex tables have been proposed. Since these tables are numerical, it is extremely important to make the table data smooth, yet preserve its convexity. Smoothness ensures smooth convergence of the convex optimizer. The existing smoothing techniques either cannot preserve convexity or suffer from high execution time. In this paper, we propose a linear time algorithm to smoothen a given numerically convex data while preserving its convexity. Our proposed algorithm *SmartSmooth* can smoothen the data in linear time and also preserve its convexity, so that it can be used in convex optimization problems. We demonstrate a 30X reduction in fitting square error over $PosynomialFit$, and a 3X reduction in fitting square error over $ConvexSmooth$. We also demonstrate faster convergence of the convex optimizer using our smoothing algorithm. $ConvexFit$ along with *SmartSmooth* can be used for generating smooth convex models for optimization of non-analytical industrial scale designs, for which fitting to analytical forms might often be too expensive.

8. 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] K. Kasamsetty, M. Ketkar and S.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] S.J. Benson and Y. 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] C.P. Chen, C.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] L. Vandenberghe and S. Boyd,"Applications of semidefinite programming", in *Applied Numerical Mathematics*, 29:283-299, 1999.

[15] S. Roy, W. Chen and C.C.P. 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.

[16] S. Roy and C.C.P. Chen,"ConvexSmooth: A simultaneous convex fitting and smoothing algorithm for convex optimization problems", in *Proceedings of the 7th International Symposium on Quality Electronic Design*, 2006.