IPSA: Integer Programming via Sparse Approximation for Efficient Test-chip Design

Qicheng Huang*, Chenlei Fang*, Zeye Liu*, Ruizhou Ding and R. D. (Shawn) Blanton*
*Advanced Chip Testing Laboratory (www.ece.cmu.edu/~actl/)
Department of Electrical and Computer Engineering
Carnegie Mellon University
Email: {qichengh, chenleif, zeyel, rding, rblanton}@andrew.cmu.edu

***Abstract*—Logic test chips are a key component of the yield learning process, which aim to investigate the yield characteristics of actual products that will be fabricated at high volume. Mathematically, the design of a logic test chip with such an objective may involve solving a constrained under-determined equation for an integer vector solution, which is unfortunately, NP-hard. Existing solving methods are not applicable due to lack of accuracy or high computational complexity. We propose a method called IPSA (Integer Programming via Sparse Approximation) to solve this integer programming (IP) problem in an effective and efficient manner. By solving a transformed sparse-regression problem and a subsequent rounding process, a solution can be achieved with comparable error to the optimal solution of the original IP problem but using far less time and memory. Experiments with seven industrial examples demonstrate that with more than 100× speed-up, IPSA achieves a similar or even better solution compared to directly solving the original problem with a commercial IP solver.**

I. Introduction

The continued scaling of integrated circuits (ICs) introduces complex interactions between layout features, which can lead to manufacturability issues that reduce yield. In recent years, foundries have increased capital expenditures [1] and time to inspect and review equipment for process control and yield stabilization/improvement. Logic test chips are a key component of the yield learning process. The main objective for a test chip is to identify systematic defects that may lead to yield loss during the high-volume production.

Fabless, IDMs (integrated device manufacturers), and foundries all use product-like designs as test chips, which are not ideal since they do not inherently optimize testability and diagnosability [2, 3]. A Logic Characterization Vehicle (LCV) [3-5] addresses the drawbacks of traditional test-chip approaches. As shown in Figure **1**(a), the basic architecture of the LCV is a two-dimensional array of functional unit blocks (FUBs). Each FUB consists of a logical circuit that implements a special FUB function (e.g., VH-bijective [3]). Leveraging C-testability theory [6], the LCV can guarantee defect detection at array primary outputs if it is detectable at the FUB boundary, which also enhances the diagnosability.

From a design perspective, an ideal test chip would perfectly incorporate the characteristics of actual customer/product designs. In other words, it is of the utmost importance that the test chip physically mimics actual designs that will go into high-volume production. Otherwise, it is possible that a test chip misses design-fabrication issues inherent to a product, or results in unnecessary and even detrimental fixes of false-alarm issues. In order to incorporate the physical characteristics of standard-cells into a LCV, one of the basic requirements is to establish the same standard-cell usage distribution derived from actual customer/product design(s) within a test chip. To this end, a set of FUB logic implementations need to be identified that altogether have a standard-cell histogram that closely matches a target distribution.

The task to properly identify FUB implementations that mimic a given standard-cell usage distribution (referred to as "LCV implementation" for the rest of paper) can be accomplished by solving a constrained under-determined equation for an integer vector solution [7, 8] (more details in Section II.A). Such an *integer programming* (IP) problem has been proved to be NP-hard [9], if each possible integral solution needs to enumerated. The *branch and bound* algorithm [10], adopted by widely-used commercial IP solvers [11, 12], reduces the computation efforts of pure enumeration by searching branches and discarding unpromising ones. However, the time and space complexity of branch and bound is still exponential in the number of variables. For the LCV implementation task, the optimization problem is extremely large (e.g., with $\sim 10^6$ variables), which greatly challenges even the best solvers. For example, a server with 64 2.2GHz CPU cores and 1TB of RAM is unable to handle an IP problem with $6{\times}10^6$ variables due to insufficient memory; even for a problem with $3{\times}10^4$ variables, it took more than one day to reach a solution with satisfactory error. Such a dilemma has become a severe bottleneck in the overall test-chip design process, especially for fabless and foundries that require fast yield ramping (e.g., a foundry may need to fabricate a new product each month). Therefore, it is crucial to develop a more efficient solver with both reduced runtime and compute resources.

Towards this goal, we propose a methodology called *IPSA* (Integer Programming via Sparse Approximation) to find a near-optimal solution for the LCV implementation problem that requires shorter runtime and less memory than the original formulation. Our work is motivated by the observation that the original IP problem can be transformed into a sparse-regression problem without the integer constraint followed by a subsequent rounding process. Compared to directly solving the original IP problem, IPSA results in similar distribution-matching error, but requires far less time and memory due to the sparse restriction. Besides this new solution approach, other contributions of this work include:

- Theoretical and intuitive explanations of why a relaxed

978-1-5386-6648-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCD46524.2019.00011

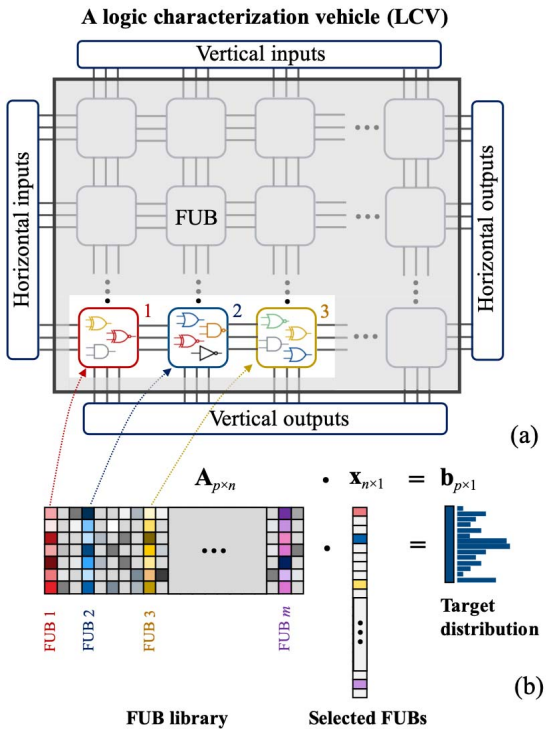

Figure 1. Optimization formulation for test-chip (LCV) implementation: (a) a structural illustration of the LCV under design and (b) mathematical formulation of the matching objective. The goal of test-chip implementation is to select a suitable combination of implementations from a FUB library described by matrix **A**, so that the overall inclusion of standard cells within the test chip mimics the target distribution described by vector **b**. The FUB implementations denoted by the colored columns are selected and their corresponding counts can be found in vector **x**, which is the objective of the solver.

sparse regression with a subsequent rounding process can result in a sub-optimal solution with comparable error to the solution derived by conventional branch-and-bound algorithm. Specifically, we analyze two sources of errors involved in the proposed methodology, and show that by finding a sparse solution, one type of error can be decreased significantly while not increasing the other type too much.

- Deployment of two strategies to efficiently solve the transformed sparse regression problem, namely: (i) greedy forward step-wise regression and (ii) L_1-regularization. Analysis of error bounds and computational cost are provided, which verify the correctness and efficiency of the two strategies.
- Application to other large-scale constrained IP problems with the aim of solving an underdetermined linear system.

The remainder of this paper is organized as follows. In Section II, we introduce the mathematical formulation of LCV implementation and review existing methods for solving the formulation. The IPSA method is presented in Section III, including how it is derived from an analysis of two types of errors and theoretical verification of correctness and efficiency. Then the efficacy of the method is demonstrated using seven case studies in Section IV. Section V concludes the paper and gives directions for future work.

II. Problem Formulation and Challenges

In this section, we first introduce the optimization problem for LCV design, and then describe the existing methods for solving it and their respective limitations.

A. The Optimization Problem for LCV Implementation

The goal of LCV implementation is to implement each FUB in the array with a proper combination of standard cells, so that all the standard cells within the entire LCV closely match a usage distribution. The distribution is usually in the form of a histogram identified during standard-cell measurement from target design(s) such as a flagship product (for fabless), or customer designs (for foundries).

However, it is quite challenging to generate a proper set of FUB implementations where (i) each implements a specific function (e.g., VH-bijective), and (ii) with an overall standard-cell usage that matches the target distribution. This is mainly the case because synthesis is not geared to create a logic implementation with constraints on the cell types and their counts. For example, it is difficult to ensure synthesis generates an implementation with only 100 INV, 50 NOR and 50 NAND gates. A workaround to address this shortcoming developed in past work [7, 8] involves synthesizing a large number of FUBs (up to a million) using different subsets of the standard-cell library. Each FUB is characterized for the types and numbers of cells it has, its testability, diagnosability, etc., and those with promising properties are kept in a *FUB library*. The FUB library is similar to a box of Legos where each piece of brick being a specific logic implementation. The goal of LCV implementation task then becomes selecting a suitable set of Legos to mimic a target (i.e., a standard-cell distribution), which can be formulated as the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{x}} \ ||\mathbf{Ax}-\mathbf{b}||_2^2 \\ &\text{s.t. } \mathbf{x}\in\mathbb{Z}^n \\ &\mathbf{Dx}\geq\mathbf{d} \end{aligned} \tag{1}$$

where $\mathbb{Z}$ denotes the integer set, $\mathbf{A}\in\mathbb{R}_+^{p\times n}$ and $\mathbf{D}\in\mathbb{R}_+^{m\times n}$ are a $p\times n$ and an $m\times n$ non-negative matrix respectively, and $\mathbf{b}\in\mathbb{R}_+^p$ and $\mathbf{d}\in\mathbb{R}_+^m$ are non-negative vectors of p and m dimensions, respectively. The symbol $\geq$ denotes the element-wise greater or equal.

As illustrated in Figure **1**(b), each column of **A** represents a FUB implementation and each row represents a specific type of standard cell. So each entry in **A** indicates the number of cell instances for the corresponding FUB implementation. The vector **x** represents the numbers of FUB instances selected to form the LCV. The target distribution is denoted by **b**, containing the required number of each standard cell. Mathematically, the columns of **A** can also be called the "basis", and the goal is to decompose the target vector **b** using the basis with **x** being the corresponding coefficients. In Eq. (1), the L_2-norm of the difference between the actual and target histograms is used

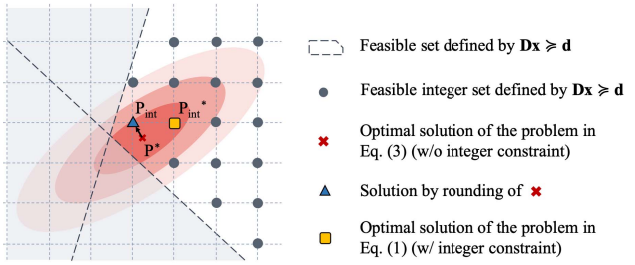

Figure 2. A two-dimensional geometric explanation for the optimization problem in Eq. (1). The points on the same contour have the same value for the objective function. By solving the original IP problem in Eq. (1), we are trying to find a point on the unshaded grid nodes that has the smallest objective value. Also, the solution derived from the relax-round strategy (solving the problem in Eq. (3) and then rounding) can be far from the true optimal integer solution.

as the minimization objective. $\mathbf{x} \in \mathbb{Z}^n$ constraints the number of selected instances of a given FUB to be an integer. Other constraints on $\mathbf{x}$ are inclueded in $\mathbf{Dx} \geq \mathbf{d}$. For example, one basic constraint is that the derived integers in $\mathbf{x}$ must be non-negative, which can be formulated by setting $\mathbf{D} = \mathbf{I}$ and $\mathbf{d} = \mathbf{0}$; if on the other hand, the practitioner has the requirement that there must exist at least some minimum for a set of standard cells, then the constraint can be reformulated by setting

$$\mathbf{D} = \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}, \mathbf{d} = \begin{bmatrix} \mathbf{0} \\ \mathbf{c} \end{bmatrix} \tag{2}$$

where $\mathbf{c}$ has the same size as $\mathbf{b}$, with each entry representing the number of instances for a given standard cell. The number of columns in $\mathbf{A}$ is significantly larger than its row count, since a more diverse FUB library provides more "Lego"s to choose from and, hence, is more flexible for matching a target distribution. It implies that the linear system $\mathbf{Ax} = \mathbf{b}$ is highly under-determined, i.e., the number of variables is much higher than the number of equations.

Figure **2** gives a two-dimensional geometric explanation for the optimization problem described in Eq. (1). The points on the same contour have the same value for the objective function in Eq. (1). The unshaded area is the feasible area defined by the linear constraints in $\mathbf{Dx} \geq \mathbf{d}$. Beacuse only an integer solution is possible due to $\mathbf{x} \in \mathbb{Z}^n$, only the points on the grid nodes make up the feasible solution set. Essentially, solving the optimization problem equates to identifying a point located at one of the unshaded grid nodes (as denoted by dark-blue dots) that corresponds to the smallest objective value.

B. Existing Methods

Solving the IP problem in Eq. (1) is unfortunately, NP-hard [9]. The most direct and naive workaround is a *relax-round* strategy that first solves a relaxed version of the problem by eliminating the integer constraint as:

$$\begin{aligned} &\min_{\mathbf{x}} ||\mathbf{Ax} - \mathbf{b}||_2^2 \\ &\text{s.t. } \mathbf{Dx} \geq \mathbf{d} \end{aligned} \tag{3}$$

and then rounding the real number solution $\mathbf{x}_{\text{real}}$ to the final integer solution $\mathbf{x}_{\text{int}}$ by a rounding function $f_{\text{round}}(\cdot)$:

$$\mathbf{x}_{\text{int}} = f_{\text{round}}(\mathbf{x}_{\text{real}}). \tag{4}$$

The meaning of $f_{\text{round}}(\cdot)$ is to map the relaxed solution to the nearest integer grid node. However, the solution resulting from this strategy can be far from optimal, because the nearest mapped integer point can have a less-desirable objective value compared to optimal solution, depending on the shape of the objective function. Figure **2** shows an example for such cases, where the mapped point P_{int} (as denoted by the blue triangle) is not necessarily the optimal integer solution $\mathrm{P}^*_{\text{int}}$ (as denoted by the yellow square). Their difference in objective function can be exaggerated when the dimension of the solution space increases.

Most commercial tools (e.g., [11, 12]), on the other hand, are based on a more accurate branch-and-bound algorithm [10, 13, 14] targeting the IP problem in Eq. (1) directly. The branch-and-bound algorithm starts with a feasible point $\mathbf{x}^{(0)}$ (i.e., a point satisfies all the constraints). The corresponding object function value $f^{(0)}$ of the best solution so far serves as the upper bound. The n dimensions of $\mathbf{x}$ lead to a searching space defined by n variables. The space is then iteratively divided to search for an optimal solution. In each iteration, one variable x_j ($j \in \{1, 2, ..., n\}$) is selected and two "branches" of sub-problems are formed. The two sub-problems have the new constraints $x_j \leq x_j^{(0)}$ and $x_j > x_j^{(0)}$ respectively, and the lower bound for each sub-problem is calculated. If the lower bound of either sub-problem is larger than the current upper bound, the sub-space it defines can be safely discarded, as the optimal value will not exist in this sub-space. By repetition of such a progress, a *search tree* keeps growing and being pruned until the optimal leaf is reached (i.e., the subspace contains only one candidate and cannot be further divided). Given sufficient time and memory, the algorithm is guaranteed to find the global-optimal solution. However, this tree-based searching process has time and memory complexity exponential to the variable dimension n, and can easily be impractical for large-scale problems.

Due to the high complexity of branch-and-bound, other works have instead searched for a sub-optimal solution that exhibits lower complexity. For example, the authors of [9] have proposed a method based on a semi-positive definite (SDP) relaxation of the original IP problem. A randomized algorithm is then applied to find a feasible solution. This algorithm has an overall complexity of $O(n^3)$, which is much lower than that of branch-and-bound. However, it assumes the integer programming problem has no other constraints, which is not the case in our application scenario. The consequence of adding the inequality constraints in Eq. (1) to the SDP problem results in long runtimes for commercial tools to solve, which is impractical for our problem considering its large scale.

III. Integer Programming via Sparse Approximation

While the relax-round strategy described in Eqs. (3)-(4) usually cannot provide a sufficiently accurate solution, the relaxed problem in Eq. (3) can be solved by convex solvers (e.g., [15]) with limited compute time as compared to branch-and-bound. IPSA is based on error analysis of the relax-

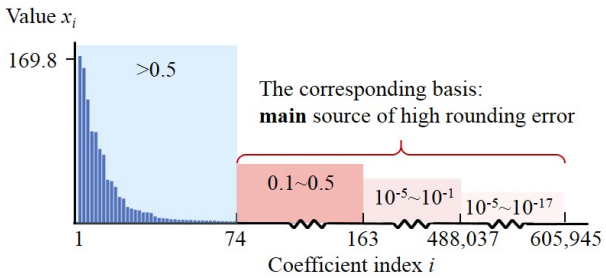

Figure 3. Solved coefficient values for Eq. (3). The solution is non-sparse but more than 99.9% of the coefficients are extremely small. The small coefficients lead to high rounding error as compared to fitting error.

round solution: by transforming Eq. (3) to a sparse-regression problem, we can utilize the fast solving speed of the relax-round strategy while decreasing the error to a comparable level as the optimal IP solution.

In this section, we first introduce two sources of errors of the relax-round strategy, namely *fitting error* and *rounding error*, and then demonstrate their relationship to the sparsity of the solution. For the transformed sparse-regression problem, we further propose two solving strategies – forward step-wise regression and L_1-regularization. Finally, detailed analysis concerning the advantages and limitations of the two strategies, as well as their error bound and computation cost will be provided to verify their correctness and efficiency.

A. Two Error Types

From a statistical view, we assume that there is an underlying optimal solution $\mathbf{x}^*$ such that $\mathbf{b} = \mathbf{A}\mathbf{x}^* + \epsilon$, where $\epsilon \in \mathbb{R}^p$ is the Gaussian noise added to each dimension of $\mathbf{b}$, i.e., $\epsilon_j \sim N(0, \sigma^2), j = 1, 2, ..., p$. For any given real-number vector $\mathbf{x}$, the difference after applying the rounding function f_{round} on it is denoted as $\Delta\mathbf{x}$. For the sake of simplicity, we assume a stochastic rounding strategy[1] so that both $\Delta\mathbf{x}$ and ϵ are random variables. The expected error of the rounded solution can be expressed as:

$$\begin{aligned}&\mathbb{E}\ \|\mathbf{A}\cdot f_{\text{round}}(\mathbf{x}) - \mathbf{b}\|_2^2 = \mathbb{E}_{\epsilon}\mathbb{E}_{\Delta\mathbf{x}}\ \|\mathbf{A}(\mathbf{x}+\Delta\mathbf{x}) - \mathbf{b}\|_2^2 \\ =&\mathbb{E}_{\epsilon}\mathbb{E}_{\Delta\mathbf{x}}\left[\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2 + 2(\mathbf{A}\mathbf{x}-\mathbf{b})^T\mathbf{A}\Delta\mathbf{x} + \|\mathbf{A}\Delta\mathbf{x}\|_2^2\right]. \end{aligned} \quad (5)$$

The expectation of the crossing term in Eq. (5) can be re-written as:

$$\begin{aligned}&\mathbb{E}_{\epsilon}\mathbb{E}_{\Delta\mathbf{x}}\left[(\mathbf{A}\mathbf{x}-\mathbf{b})^T\mathbf{A}\Delta\mathbf{x}\right] \\ =&\mathbb{E}_{\epsilon}\Big[(\mathbf{A}\mathbf{x}-\mathbf{b})^T\mathbf{A}\mathbb{E}_{\Delta\mathbf{x}}\big[\Delta\mathbf{x}|\mathbf{x}\big]\Big] = 0, \end{aligned} \quad (6)$$

where we use the fact that $\mathbb{E}_{\Delta\mathbf{x}}\big[\Delta\mathbf{x}|\mathbf{x}\big] = 0$ as implied by the stochastic rounding assumption. Substituting Eq. (6) into Eq. (5) yields:

$$\mathbb{E}\ \|\mathbf{A}\mathbf{x}_{\text{int}} - \mathbf{b}\|_2^2 = \mathbb{E}\ [\ \underbrace{\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2}_{\text{fitting error}} + \underbrace{\|\mathbf{A}\Delta\mathbf{x}\|_2^2}_{\text{rounding error}}\ ] \quad (7)$$

[1]We use stochastic rounding [16] for simplicity of analysis, but the similar arguments about the error composing of fitting and rounding errors generally hold if using nearest-rounding.

The error is composed of two parts: fitting error which measures how well $\mathbf{x}$ can reconstruct $\mathbf{b}$, and rounding error which indicates how far $\mathbf{x}$ is from an integer point. Suppose there are N non-zeros elements in $\mathbf{x}$, an upper bound on the rounding error can be calculated by

$$\begin{aligned}\|\mathbf{A}\Delta\mathbf{x}\|_2^2 &= \left\|\sum_j \mathbf{A}_j\Delta x_j\right\|_2^2 \le \sum_j \left\|\mathbf{A}_j\Delta x_j\right\|_2^2 \\ &\le N\cdot\max_j \|\mathbf{A}_j\|_2^2, \end{aligned} \quad (8)$$

where we use the fact that the maximum possible stochastic rounding error is 1. Since the upper bound in Eq. (8) is proportional to N, decreasing the number of non-zero elements in $\mathbf{x}$ reduces the rounding error. But on the other hand, a more sparse $\mathbf{x}$ implies fewer types of FUB implementations are used (i.e., fewer basis to decompose $\mathbf{b}$ onto), which increases the fitting error.

Despite the trade-off between rounding and fitting errors with respect to N, the impact of N on rounding error is much higher, especially when N is not very small (e.g., N >100). By studying various cases of the the relaxed IP problem in Eq. (3), we find the solutions have two important characteristics: (i) they are not sparse, i.e., almost all of the elements in $\mathbf{x}$ are non-zero; (ii) most elements in $\mathbf{x}$ are extremely small (e.g., more than 99.9% are less than 0.5). Figure **3** shows an example by plotting the values of all the elements in $\mathbf{x}$. Only 74 out of 605,945 coefficients are larger than 0.5 and a fairly large proportion of the coefficients are even smaller than 10^{-5}. Studying the contribution from the basis with small coefficients to the fitting/rounding error reveals that such basis bring much higher rounding error than their contribution to reducing the fitting error. If we divide $\mathbf{x}$ into two vectors $\mathbf{x}_{\text{L}}$ (coefficients larger than 0.5, with corresponding basis $\mathbf{A}_{\text{L}}$) and $\mathbf{x}_{\text{S}}$ (smaller than 0.5, with basis $\mathbf{A}_{\text{S}}$), then the difference in fitting error and rounding error brought by basis in $\mathbf{A}_{\text{S}}$ can be defined as follows, respectively:

$$\Delta\epsilon_f = \left(\|\mathbf{A}_{\text{L}}\cdot\mathbf{x}'_{\text{L}} - \mathbf{b}\|_2 - \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2\right)\ /\ \|\mathbf{b}\|_2 \quad (9)$$

$$\Delta\epsilon_r = \|\mathbf{A}_{\text{S}}\cdot\Delta\mathbf{x}_{\text{S}}\|_2\ /\ \|\mathbf{b}\|_2 \quad (10)$$

where $\mathbf{x}'_{\text{L}}$ are the coefficients solved from Eq. (3) by using only the basis in $\mathbf{A}_{\text{L}}$, and $\Delta\mathbf{x}_{\text{S}}$ is calculated by:

$$\Delta\mathbf{x}_{\text{S}} = \mathbf{x}_{\text{S}} - f_{\text{round}}(\mathbf{x}_{\text{S}}) \quad (11)$$

In the case shown in Figure **3**, $\Delta\epsilon_f = 0.0149$ while $\Delta\epsilon_r = 0.1446$, which means that considering the basis in $\mathbf{A}_{\text{S}}$ brings 10× more rounding error than their contribution to reducing the fitting error. Thus, if $\mathbf{A}_{\text{S}}$ can be identified beforehand and eliminated from $\mathbf{A}$, then the total error of the solution would be significantly reduced.

B. Sparse Regression and Solving Strategies

Based on analysis of the two types of errors, we transform Eq. (3) to a sparse-regression problem, namely:

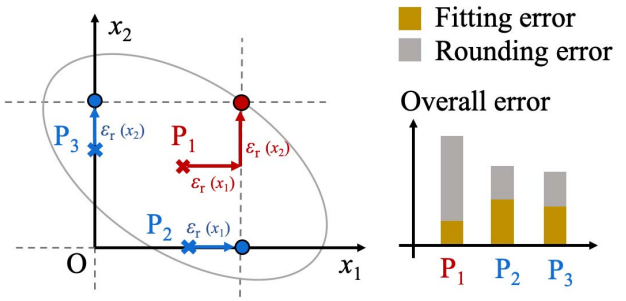

Figure 4. P_1~P_3 are three solutions of the relaxed problem in Eq. (3). P_1 is has the least fitting error but higher rounding error because it is not sparse. By sparse-regression in Eq. (12), we are trying to find solutions like P_2 and P_3, both of which have lower overall error.

$$\begin{aligned} &\min_{\mathbf{x}} ||\mathbf{Ax}-\mathbf{b}||_2^2 \\ &\text{s.t. } \mathbf{Dx} \geq \mathbf{d} \\ &\quad ||\mathbf{x}||_0 \leq \lambda \end{aligned} \tag{12}$$

The L_0-norm of $\mathbf{x}$ in Eq. (12) restricts the number of non-zero elements below a pre-defined hyper-parameter λ. By solving Eq. (12), we are expecting to decrease N for small rounding error and at the same time find a suitable set of basis so that the fitting error can also be under control.

Note that the regression problem is highly under-determined, implying that the number of basis in $\mathbf{A}$ is so large that many of them are very similar. Thus, for Eq. (3), it is probable to find many sub-optimal points near the optimal solution. In Eq. (12) we aim to find such sub-optimal points that are on the axis, which have slightly higher fitting error but far lower rounding error. For example, as the illustration shown in Figure **4**, P_1~P_3 are three solutions of Eq. (3). P_1 is the optimal solution with the least fitting error, but during the rounding process, it has rounding errors in every dimension, so its totally error is higher than other two sub-optimal solutions which have smaller rounding error due to sparsity. By solving Eq. (12), we are trying to find solutions like P_2 and P_3, which are sub-optimal in terms of fitting error but with much lower rounding and overall errors. Since Eq. (12) is NP-hard due to the non-convexity of L_0-norm, two alternative strategies are proposed.

Solving Strategy 1: forward step-wise regression

When no other constraints are specified besides $\mathbf{x} \geq \mathbf{0}$, a greedy forward step-wise strategy can be adopted to iteratively select a subset of the basis that can significantly decrease fitting error. As summarized in Algorithm 1, we begin with a residual $\mathbf{r}^{(0)} = \mathbf{b}$ and an empty basis index set V (step 1). In each iteration, a basis that is most positively correlated with the residual is identified by evaluating the normalized inner product defined in Eq. (13). We need to ensure that the inner product is non-negative to ensure the coefficients with respect to the selected basis satisfy the constraint $\mathbf{x} \geq \mathbf{0}$. So in step 3, if no more basis are positively correlated to the residual, the iteration terminates. Once a new basis is chosen, its index is added to V and the regression problem is re-solved based on the current selected basis in step 5 (which is easily solvable by convex solvers such as [15]). A new $\mathbf{r}^{(k)}$ is updated in step 6, indicating the residual that the remaining basis needs to fit. If not terminated at step 3, an iteration will terminate when its time reaches a pre-defined threshold $N_{\max}$. Finally, all the coefficients with respect to the un-selected basis are set to zero (step 7).

Algorithm 1 Forward Step-wise Regression

Input: Objective $\mathbf{b}$, the basis matrix $\mathbf{A}$ and the maximum number of iterations $N_{\max}$
Output: Coefficients $\mathbf{x}$.
1. Initialize: $\mathbf{r}^{(0)} = \mathbf{b}$, the index set $V = \varnothing$.
for $k = 1, \ldots, N_{max}$ **do**
  2. Find the index i so that

$$i = \text{argmax}_{j\in\{1,\ldots,n\}} \left\langle \mathbf{r}^{(k-1)}, \frac{\mathbf{A}_j}{|\mathbf{A}_j|} \right\rangle, \tag{13}$$

  3. **if** $\left\langle \mathbf{r}^{(k-1)}, \frac{\mathbf{A}_i}{|\mathbf{A}_i|} \right\rangle \leq 0$ **then**
    break
  end
  4. Update $V = V \cup \{i\}$.
  5. Solve the following optimization problem:

$$\min_{\mathbf{x}^{(k)}} \left\| \sum_{j\in V} \mathbf{A}_j x_j - \mathbf{b} \right\|_2^2 \tag{14}$$

$$\text{s.t. } x_j \geq 0, \quad j \in V \tag{15}$$

  for solution $\hat{\mathbf{x}}^{(k)} = \{\hat{x}_j, j \in V\}$.
  6. Update the residual by $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{b}^{(k)}, \mathbf{b}^{(k)} = \sum_{j\in V} \mathbf{A}_j \hat{x}_j$
end
7. Set the un-selected coefficients to 0: $x_j = 0, j \notin V$

Solving Strategy 2: L_1-regularization

If additional constraints are required besides $\mathbf{x} \geq \mathbf{0}$ (e.g., Eq. (2)), Strategy 1 cannot be used because selecting a basis during iterations makes it difficult to ensure the final solution satisfy the additional constrains. In such cases, L_1-regularization can be used to solve the constrained sparse-regression problem.

The main idea of L_1-regularization is to relax Eq. (12) by replacing the L_0-norm term with L_1-norm:

$$||\mathbf{x}||_1 \leq \lambda. \tag{16}$$

The L_1-norm of a vector is defined as the summation of the absolute value of all the elements, which is a convex function. Various theoretical studies from the statistics community demonstrate that under some general assumptions both L_0-norm regularization and L_1-norm regularization result in the same solution [17].

The two strategies have their respective advantages and limitations. Strategy 1 is more heuristic but works very well in practice. In each iteration, the optimization problem in Eqs. (14)-(15) contains only a small number of variables so that it can be efficiently solved. In addition, with Strategy 1, selection of the hyper-parameter λ can be avoided, because

in practice we can integrate the rounding process and track the error in each iteration by rounding $\hat{\mathbf{x}}^{(k)}$ and calculating the residual. Finally, we can simply select the best result within all the iterations (an example will be given in Section IV). The downside of Strategy 1 however is that it can only solve the problem with constraint $\mathbf{x} \geq \mathbf{0}$, which highlights the strength of Strategy 2 – flexibility to deal with any linear constraints. However, Strategy 2 requires careful tuning of the hyper-parameter λ, which significantly increases the total runtime if aiming for an optimal λ.

C. Error Bound and Computational Cost

Here we analyze the error bound of the two strategies.

For Strategy 1, without loss of generality, we assume that the object $\mathbf{b}$ can be constructed through a non-negative linear combination of basis $\mathcal{D} = \{\psi_1, \psi_2, ..., \psi_n\}$, where $\psi \in \mathbb{R}^p, \|\psi\|_2 = 1$ for all $\psi \in \mathcal{D}$. This dictionary $\mathcal{D}$ can be constructed by normalizing each column of matrix $\mathbf{A}$. In other words, we assume that there exists an underlying set of coefficients $\mathbf{x}^*$ that $\mathbf{b} = \sum_j x_j^* \psi_j$. Based on the proof in the appendix, we have the following theorem concerning the error bound of the residual $\mathbf{r}^{(k)}$ of step k:

Theorem 1. *Let $\|\mathbf{b}\|_{\mathcal{L}_1} = \sum_j x_j^* < \infty$, the residual $\mathbf{r}^{(k)}$ after k steps satisfies*

$$\|\mathbf{r}^{(k)}\|_2 \leq \frac{\|\mathbf{b}\|_{\mathcal{L}_1}}{\sqrt{k+1}} \tag{17}$$

for all $k \geq 1$.

Theorem 1 demonstrates, using Algorithm 1, the residual-error bound decreases at the rate of $O(k^{1/2})$, which assures error convergence of Strategy 1 (i.e., the fitting error). On the other hand, after the subsequent rounding process, the rounding error increases approximately at the rate of $O(k)$ (since the upper-bound is proportional to N). Considering the trade-off between the two parts of error, there should exist an optimal k that minimizes overall error.

For Algorithm 2, with the same "$\mathbf{b} = \mathbf{A}\mathbf{x}^* + \epsilon$" assumption as stated at the beginning of Section III.A, and that $\|\mathbf{A}_j\|_2^2 \leq n$, for $j = 1, ..., p$, we have the following theorem (see the appendix for the proof of Theorem 2):

Theorem 2. *Set $\lambda = \|\mathbf{x}^*\|_1$ in Eq.* (16), then with probability at least $1 - \delta$, the expectation of error between the fitted result $\mathbf{A}\hat{\mathbf{x}}$ and $\mathbf{A}\mathbf{x}^*$ has the error bound

$$\frac{1}{p}\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{A}\mathbf{x}^*\|_2^2 \lesssim \|\mathbf{x}^*\|_1 \sqrt{\frac{\log n}{p}}. \tag{18}$$

It shows that the error bound is proportional to the L_1-norm of the underlying "real" coefficients $\mathbf{x}^*$. If $\mathbf{x}^*$ has a small L_1-norm value, the solution $\hat{\mathbf{x}}$ given by Strategy 2 will not have a large fitting error.

Now we demonstrate that the two strategies have polynomial complexity, which is a significant speed-up over branch-and-bound algorithm's exponential complexity.

For Strategy 1, step 2 in Algorithm 1 involves n vector multiplications of length p, which means $O(np)$ floating-point operations (FLOPs). Step 5 is the most complex, including solving for a constrained quadratic optimization problem. Conventional commercial tools use an interior-point method that requires $O(n^2p)$ FLOPs. Step 6 is essentially a matrix-vector product and a vector subtraction, which requires $O(kp)$ FLOPs. Assuming that iteration terminates after K iterations, the overall complexity of Strategy 1 is $O(Kn^2p)$. The main calculation of Strategy 2 is to solve the L_1 regularized problem. This step can also performed by commercial tools using an interior-point method, which has $O(n^2p)$ complexity.

IV. Experiments

In this section, we apply IPSA to the LCV implementation problems, and compare the performance with the naive relax-round method, and a commercial IP solver. In addition, further analysis is conducted to demonstrate details of the method and illustrate the trade-off between fitting and rounding errors as discussed in Section III.A. All experiments are completed using a server with 64 2.2GHz CPU cores and 1TB of RAM.

A. Comparison of Methods

We compare the performance of four methods for LCV implementation: (i) IPSA with the forward step-wise strategy, (ii) IPSA with L_1-regularization, (iii) naive relax-round method, and (iv) a commercial IP solver [11]. Seven examples (denoted as Designs 1-7) with respect to real industrial test-chip design cases are used to evaluate the performance. Each example corresponds to a standard-cell library and a target standard-cell histogram. In other words, Eq. (1) is solved using specific matrix $\mathbf{A}$ and vector $\mathbf{b}$ for each design. The $\mathbf{A}$ matrices for Designs 1-3 contain 605,945 columns, indicating the solution $\mathbf{x}$ has a dimension of 605,945. The number for Designs 4-5 is 829,228, and 222,698 for Designs 6-7. A commercial convex optimization tool [15] is used for solving the relaxed problem in Eq. (3), forward step-wise regression (step 5 in Algorithm 1), and L_1-regularization.

Two metrics are used to evaluate the four solution approaches, namely standard-cell histogram matching, and runtime. Histogram matching for a specific solution $\hat{\mathbf{x}}$ is defined as follows:

$$\text{matching error} = \frac{\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_1}{\|\mathbf{b}\|_1}. \tag{19}$$

Here we use the L_1-norm because it measures the histogram mismatch in a more intuitive way than L_2-norm, while in the optimization problem we used the latter because it is more smooth and strictly convex, making it easier to solve by commercial convex optimization tools.

First we consider a scenario where the only linear constraint is $\mathbf{x} \geq \mathbf{0}$, i.e., the practitioner has no specific requirement for the least number of specific standard cell(s) in the design. In this case, both forward step-wise and L_1-regularization strategies can be applied. The two metrics of the four methods are listed in Table I, as shown in the "Overall error" and "Time" columns. For each design, the method that gives the smallest error is highlighted with bold text in each row. For

Table I
HISTOGRAM MISMATCH (ERROR) AND RUNTIME COMPARISON OF DIFFERENT METHODS.

Design	Relax-round			IPSA Forward step-wise regression			IPSA L_1-regularization			IP Solver [11]	
	Full prec. error	Overall error	Time (sec.)	Full prec. error	Overall error	Time (sec.)	Full prec. error	Overall error	Time (sec.)	Overall error	Time (sec.)
1	0.0956	0.096	404.1	0.0564	**0.0561**	549.7	0.0956	0.0951	2,595	0.086	86,000
2	0.0513	0.0517	403.1	0.0312	**0.0313**	630	0.0514	0.0514	2,625	0.035	86,000
3	0.0721	0.072	345.3	0.0466	**0.0465**	537.1	0.0718	0.0718	2,750	0.087	86,000
4	0.0056	0.0747	347.3	0.0058	0.0077	637.3	0.0056	0.0091	7,520	**0.006**	86,000
5	0.0062	0.1531	350.2	0.00089	0.0031	516.5	0.00089	0.0059	10,885	**0.001**	86,000
6	0.1158	0.1159	67.3	0.1158	**0.1153**	136.6	0.1158	0.1159	3,948	0.1159	10,601
7	0.1138	0.1147	100.5	0.1138	**0.1126**	182.9	0.1138	0.1133	6,835	0.114	7,974

Table II
HISTOGRAM MISMATCH (ERROR) AND TIME COMPARISON OF DIFFERENT METHODS WITH LINEAR CONSTRAINTS.

Design	Relax-round			IPSA L_1-regularization			IP Solver [11]	
	Full prec. error	Overall error	Time (sec.)	Full prec. error	Overall error	Time (sec.)	Overall error	Time (sec.)
4	0.0056	0.0982	547.1	0.0056	0.0067	25,651	**0.0062**	86,000
5	0.0009	0.1832	350.2	0.0068	0.0172	20,198	**0.0070**	86,000

more straight-forward comparison, the error comparison with respect to the seven designs is also plotted in Figure **5**.

As expected, the relax-round method has the highest error for almost all cases. The IP solver is supposed to have the least error theoretically but for Design 3 it has the highest. This is mainly because (i) we restrict the maximum runtime to one day due to the time limit of the server, (ii) we only allow it to use basis with 30,000 largest coefficients derived from solving Eq. (3) because server memory cannot handle a search space of the 10^5 dimensions exhibited by Eq. (1). Forward step-wise regression significantly reduces the error over 50× compared to relax-round method and in five cases it even surpasses the IP solver. L_1-regularization does not exhibit similar improvement in some cases, probably because the time limit we set does not allow a thorough exploration of the hyper-parameters. If a suitable λ is found, L_1-regularization is also expected to significantly reduce error for Designs 4 and 5 as well. For Designs 6 and 7, because the least-accurate relax-round method achieves a similar error to the IP solver, it is expected that little improvement brought can be achieved by these two sparse-regression strategies.

In terms of runtime, the relax-round method is the fastest, because it solves the simplest problem. In contrast, the IP solver is the slowest due to its high complexity; in some cases, it does not reach the final solution within the 86,000-second time limit. Forward step-wise regression runs slightly longer but comparable to relax-round, and more than 100× faster than the IP solver on average. L_1-regularization has a longer runtime, because the L_1-regularized problem has to be solved multiple times with different hyper-parameters λ to find the optimal value.

Except for the IP solver, the other three methods first solve a relaxed problem without an integer constraint, and then round the full-precision solution $\mathbf{x}_{\text{real}}$. So in addition to the final error, the errors of $\mathbf{x}_{\text{real}}$ for the three methods are also recorded and listed in columns named "Full prec. error" in

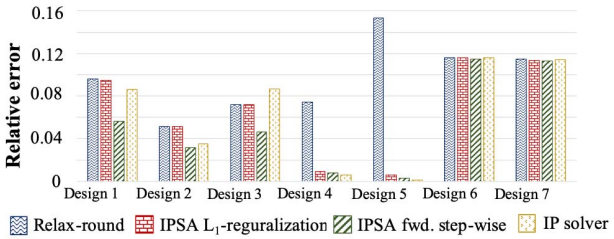

Figure 5. The comparison of the relative error (i.e., $\|\mathbf{A}\hat{\mathbf{x}}-\mathbf{b}\|_1/\|\mathbf{b}\|_1$) of the solution $\hat{\mathbf{x}}$ from each of the four methods for seven designs.

Table I. Comparing these errors with the corresponding values in "Overall error" columns, we can observe the effect of the rounding process for each method. The relax-round method is most susceptible to rounding. When its $\mathbf{x}_{\text{real}}$ is close to a integer grid node, the error resulting from the rounding process is small (e.g., for Design 6), but for other cases, Designs 4 and 5 for example, the rounding process can increase the error by 13× and 25×, respectively. In contrast, forward step-wise regression and L_1-regularization are not affected that much, because the differences between the two columns are relatively small and stable. This demonstrates that finding a sparse solution can reduce rounding error.

We also consider another scenario where the practitioner requires that the LCV implementation includes at least 20 instances for each type of standard-cells (i.e., the linear constraint becomes like Eq. (2) with elements in $\mathbf{c}$ equated to 20). In this case, forward step-wise regression is not applicable, so in Table II, we list the performance of the remaining three methods to Designs 4 and 5. Similar comparison results as Table I can be observed, that is, L_1-regularization method achieves similar error as the IP solver (much lower than relax-round) but with significantly reduced runtime.

B. Detailed Analysis

In this sub-section, the details of forward step-wise regression including an analysis of relax-round are presented to

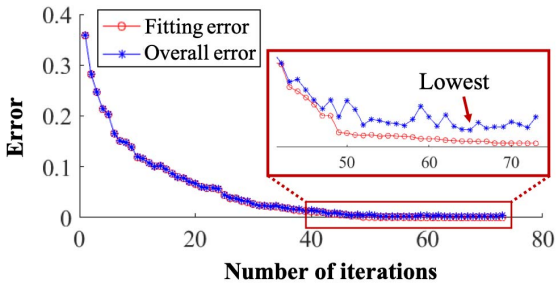

Figure 6. The change of relative fitting error (i.e., $||\mathbf{A}\hat{\mathbf{x}}_{\text{real}} - \mathbf{b}||_1/||\mathbf{b}||_1$) and relative overall error (i.e., $(||\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}||_1)/||\mathbf{b}||_1$) as the iterations of Algorithm1 proceeds.

demonstrate the trade-off between these two sources of errors.

For forward step-wise regression, the rounding process can be integrated into each iteration so that the final error can be tracked across iterations. In this way, the searching for hyper-parameters can be eliminated, since we can simply choose the one with least overall error as the final solution. For example, Figure **6** shows the change of (i) fitting error with the full-precision solution $\mathbf{x}_{\text{real}}$ and (ii) overall error with the rounded solution $\hat{\mathbf{x}}$, as the iterations proceeds for Design 5. Because in each iteration, a new basis is included in the candidate set that $\mathbf{b}$ decomposes onto, the fitting error continues to decrease. Generally, the overall error is close to the fitting error because the effects of rounding are not obvious for the sparse solution. However, as the number of non-zero coefficients increases, the gap between the two curves becomes obvious and the overall error starts fluctuating due to the varying rounding errors. The lowest overall error is achieved at the 65-th iteration instead of the last one, and we take the solution at the 65-th iteration as the final solution.

Another experiment is conducted to demonstrate the trade-off between fitting and rounding error as the sparsity of the solution varies. For the relax-round method, we use only a subset of basis, randomly selected from the columns of $\mathbf{A}$ for Design 5 and gradually include more basis to match the same target distribution. Figure **7** shows the change of fitting error and overall error with respect to different sizes of the basis (N). The deviation of the two curves is the results from the rounding error. When N is small, the effect of rounding is negligible. However, when N grows larger than 2×10^5, the rounding error begins to dominate the fitting error, thus demonstrating the trade-off between them as a function of solution sparsity. The optimal sparsity is determined by tracking of errors during iterations of forward step-wise regression and tuning hyper-parameter λ for L_1-regularization.

V. Conclusions

Incorporation of physical characteristics from actual products is the utmost important requirement for test-chip design. One of the design objectives is matching a target standard-cell usage distribution – essentially a large-scale IP problem, which presents significant challenges to existing solution techniques that lead to inaccuracy or high demand for compute resources. In this paper we describe a method called IPSA to solve

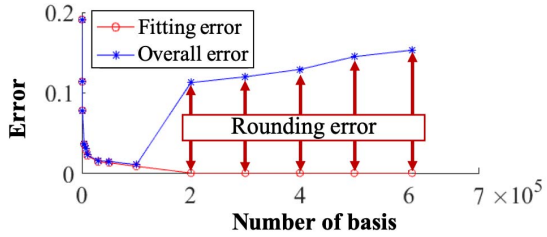

Figure 7. The change of relative fitting error (i.e., $||\mathbf{A}\hat{\mathbf{x}}_{\text{real}} - \mathbf{b}||_1/||\mathbf{b}||_1$) and relative overall error (i.e., $(||\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}||_1)/||\mathbf{b}||_1$) of the relax-round strategy, based on different basis size (N).

the IP problem in an effective way – solving a transformed sparse-regression problem with a subsequent rounding process. Two strategies for solving sparse-regression, namely forward step-wise regression and L_1-regularization, are investigated for solving the sparse-regression problem. The former has generally better performance and the latter has more flexibility. Experiments demonstrate that compared to directly solving the original problem with a commercial IP solver, IPSA achieves a similar or even better solution with more than 100× speed-up.

Besides the test-chip design application in this paper, IPSA can also be applied to other large-scale IP problems with the aim of solving a linear system with linear constraints. Especially, IPSA is expected to achieve high efficiency when the linear system is highly under-determined and the underlying optimal solution is sparse. Future work may include exploration of other design objectives other than standard-cell distribution matching by effectively solving optimization problems with different objective functions.

VI. Appendix

A. *Proof of Theorem* 1

Proof. For proof simplicity, we denote the vector selected in Eq. (13) in Algorithm 1 as $\mathbf{g}^{(k)}$. In other words:

$$\mathbf{g}^{(k)} = \text{argmax}_{\psi \in \mathcal{D}} \langle \mathbf{r}^{(k-1)}, \psi \rangle. \tag{20}$$

Given the facts that (i) $\mathbf{b}^{(k)}$ is the best approximation to $\mathbf{b}$ from $\text{Span}(\{\psi_j, i \in V^{(k)}\})$, and (ii) the approximation of $\mathbf{r}^{(k-1)}$ from the set $\{a \cdot \mathbf{g}^{(k)} : a \in \mathbb{R}\}$ is $\langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)} \rangle \mathbf{g}^{(k)}$, we have $\|\mathbf{b} - \mathbf{b}^{(k)}\|^2 \leq \|\mathbf{b} - \mathbf{b}^{(k-1)} - \langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)} \rangle \mathbf{g}^{(k)}\|^2$. Because $\mathbf{r}^{(j)} = \mathbf{b} - \mathbf{b}^{(j)}, j = 1, ..., k$, the following holds:

$$\begin{aligned} \|\mathbf{r}^{(k)}\|_2^2 &\leq \|\mathbf{r}^{(k-1)} - \langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)} \rangle \mathbf{g}^{(k)}\|_2^2 \\ &= \|\mathbf{r}^{(k-1)}\|_2^2 - |\langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)} \rangle|^2. \end{aligned} \tag{21}$$

Using the equations $\mathbf{b} = \mathbf{b}^{(k-1)} + \mathbf{r}^{(k-1)}$ and $\langle \mathbf{b}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle = 0$, we have

$$\begin{aligned} \|\mathbf{r}^{(k-1)}\|^2 &= \langle \mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle = \langle \mathbf{r}^{(k-1)}, \mathbf{b} \rangle - \langle \mathbf{b}^{(k-1)}, \mathbf{r}^{(k-1)} \rangle \\ &= \langle \mathbf{r}^{(k-1)}, \mathbf{b} \rangle = \sum_j x_j \langle \mathbf{r}^{(k-1)}, \psi_j \rangle \\ &\leq \sup_{\psi \in \mathcal{D}} \langle \mathbf{r}^{(k-1)}, \psi \rangle \sum_j x_j = \sup_{\psi \in \mathcal{D}} \langle \mathbf{r}^{(k-1)}, \psi \rangle \|\mathbf{b}\|_{\mathcal{L}_1} \\ &= \langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)} \rangle \|\mathbf{b}\|_{\mathcal{L}_1}. \end{aligned} \tag{22}$$

Substituting Eq. (22) into Eq. (21) yields:

$$\|\mathbf{r}_k\|^2 \le \|\mathbf{r}^{(k-1)}\|^2\Big(1 - \frac{\|\mathbf{r}^{(k-1)}\|^2\langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k-1)}, \mathbf{g}^{(k)}\rangle^2\|\mathbf{b}\|^2_{\mathcal{L}_1}}\Big)$$
$$= \|\mathbf{r}^{(k-1)}\|^2\Big(1 - \frac{\|\mathbf{r}^{(k-1)}\|^2}{\|\mathbf{b}\|^2_{\mathcal{L}_1}}\Big). \quad (23)$$

Assume a series of non-negative numbers $a^{(0)} \ge a^{(1)} \ldots \ge a^{(k)}$, where $a^{(0)} \le M$, and $a^{(k)} \le a^{(k-1)}(1 - a^{(k-1)}/M)$, by deduction we can derive $a^{(k)} \le M/(k+1)$. Applying this to Eq. (23), and because $M = \|\mathbf{b}\|^2_{\mathcal{L}_1}$, we have:

$$\|\mathbf{r}_k\|^2 \le \frac{\|\mathbf{b}\|^2_{\mathcal{L}_1}}{k+1}. \quad (24)$$

Theorem 1 can then be derived by taking square root of each side of Eq. (24). □

B. Proof of Theorem 2

Proof. Because $\hat{\mathbf{x}}$ is the solution to Eq. (12) with the constraint Eq. (16), the following holds:

$$\|\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\|_2^2 \le \|\mathbf{b} - \mathbf{A}\mathbf{x}^*\|_2^2. \quad (25)$$

After rearranging and using Holder's inequality [18] and the bound for L_1-regularization $\lambda = \|x^*\|_1$, we have

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{A}\mathbf{x}^*\|_2^2 \le 2\langle \epsilon, \mathbf{A}\hat{\mathbf{x}} - \mathbf{A}\mathbf{x}^*\rangle = 2\langle \mathbf{A}^T\epsilon, \hat{\mathbf{x}} - \mathbf{x}^*\rangle$$
$$\le 2\|\hat{\mathbf{x}} - \mathbf{x}^*\|_1\|\mathbf{A}^T\epsilon\|_\infty \le 4\|\mathbf{x}^*\|_1\|\mathbf{A}^T\epsilon\|_\infty. \quad (26)$$

In Eq. (26), $\|\mathbf{A}^T\epsilon\|_\infty = \max_{j=1,\ldots,n}|\mathbf{A}_j^T\epsilon|$ is a maximum of n Gaussian random variables. By standard maximal inequality for Gaussian random variables, for any $\delta > 0$, with probability of at least $1 - \delta$,

$$\max_{j=1,\ldots,n}|\mathbf{A}_j^T\epsilon| \le \sigma\sqrt{2p\log(en/\delta)}. \quad (27)$$

Substituting Eq. (27) into Eq. (26), we have

$$\frac{1}{p}\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{A}\mathbf{x}^*\|_2^2 \le 4\sigma\|\mathbf{x}^*\|_1\sqrt{\frac{2\log(en/\delta)}{p}}, \quad (28)$$

which is the same as the rate given in Eq. (18). □

References

[1] R. C. Leachman and S. Ding, "Excursion Yield Loss and Cycle Time Reduction in Semiconductor Manufacturing," *IEEE Transactions on Automation Science and Engineering*, 2011.

[2] K. Y. Cho, S. Mitra, and E. J. McCluskey, "Gate exhaustive testing," in *IEEE International Conference on Test*, 2005.

[3] R. D. Blanton, B. Niewenhuis, and C. Taylor, "Logic Characterization Vehicle Design for Maximal Information Extraction for Yield Learning," in *IEEE International Test Conference*, 2014.

[4] P. Fynan, Z. Liu, B. Niewenhuis, S. Mittal, M. Strajwas, and R. D. Blanton, "Logic Characterization Vehicle Design Reflection via Layout Rewiring," in *IEEE International Test Conference*, 2016.

[5] B. Niewenhuis and R. D. Blanton, "Efficient built-in self test of regular logic characterization vehicles," in *IEEE VLSI Test Symposium (VTS)*, 2015.

[6] A. D. Friedman, "Easily testable iterative systems," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1061–1064, 1973.

[7] R. D. Blanton, B. Niewenhuis, and Z. Liu, "Design Reflection for Optimal Test Chip Implementation," in *IEEE International Test Conference*, 2015.

[8] Z. Liu, B. Niewenhuis, S. Mittal, and R. D. Blanton, "Achieving 100% Cell-aware Coverage by Design," in *Design, Automation& Test in Europe*, 2016.

[9] J. Park and S. Boyd, "A semidefinite programming method for integer convex quadratic minimization," *Optimization Letters*, vol. 12, no. 3, pp. 499–518, 2018.

[10] J. Clausen, "Branch and bound algorithms-principles and examples," *Department of Computer Science, University of Copenhagen*, pp. 1–30, 1999.

[11] N. V. Sahinidis, "BARON 18.5.9: Global Optimation of Mixed-integer Nonlinear Programs." Users manual, 2018.

[12] "GUROBI optimization." https://http://www.gurobi.com/.

[13] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM review*, vol. 33, no. 1, pp. 60–100, 1991.

[14] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," *Handbook of applied optimization*, vol. 1, pp. 65–77, 2002.

[15] M. Grant and S. Boyd, "Cvx: Matlab software for disciplined convex programming, version 2.1," 2014.

[16] "Stochastic Rounding." https://en.wikipedia.org/wiki/Rounding.

[17] E. J. Candès *et al.*, "Compressive sampling," in *Proceedings of the international congress of mathematicians*, vol. 3. Madrid, Spain, 2006, pp. 1433–1452.

[18] "Holder's inequality." https://en.wikipedia.org/wiki/Holdersinequality.