Sparse Coding for Specification Mining and Error Localization

Wenchao Li and Sanjit A. Seshia

University of California at Berkeley

Abstract. Formal specifications play a central role in the design, verification, and debugging of systems. This paper presents a new viewpoint to the problem of mining specifications from simulation or execution traces of reactive systems. The main application of interest is to localize faults to sections of an error trace we term *subtraces*, with a particular focus on digital circuits. We propose a novel sparse coding method that extracts specifications in the form of *basis subtraces*. For a set of finite subtraces each of length p, each subtrace is decomposed into a *sparse* Boolean combination of only a small number of basis subtraces of the same dimension. We formally define this decomposition as the *sparse Boolean matrix factorization problem* and give a graph-theoretic algorithm to solve it. We formalize a sufficient condition under which our approach is sound for error localization. Additionally, we give experimental results demonstrating that (1) we can mine useful specifications using our sparse coding method, and (2) the computed bases can be used to do simultaneous error localization and error explanation.

1 Introduction

Formal specifications play a central role in system design. They can serve as high-level requirements from which a system is to be synthesized. They can encode key properties that the system must exhibit, finding use in formal verification, testing and simulation. Formal specifications are also valuable as contracts for use in code maintenance and compositional design. Finally, they are also useful in debugging and error localization, in the following way: if several local properties are written for a system, covering each of its components, then a failing property can provide information about the location of the bug. It is this last application of formal specifications — for error localization and debugging in *reactive systems* — that is the main focus of this paper.

Unfortunately, in practice, comprehensive formal specifications are rarely written by human designers. It is more common to have instead a comprehensive test suite used during simulation or testing. There has therefore been much interest in automatically deriving specifications from simulation or execution traces (e.g. [8, 2]). It is important to note that, until they are formally verified, the properties generated from traces are only *likely* specifications or *behavioral signatures* of a design.

Different kinds of formal specifications provide different tradeoffs in terms of ease of generation from traces, generality, and usefulness for error localization. Büchi automata [4] provide a very general formalism, and are typically inferred by learning a finite automaton from finite-length traces and interpreting it over infinite traces. However, such automata tend to overfit the traces they are mined from, and do not generalize well to unseen traces — i.e., they are very sensitive to the choice of traces $\mathcal{T}$ they are mined from and can easily exclude valid executions outside of the set $\mathcal{T}$. Linear temporal logic (LTL) formulas [18] are an alternative. One typically starts with templates for common temporal logic formulas and learns LTL formulas that are consistent with a set of traces. If the templates are chosen carefully, such formulas can generalize well

to unseen traces. However, the biggest challenge is in coming up with a suitable set of templates that capture all relevant behaviors.

In this paper, we introduce a third kind of formal specification, which we term as *basis subtraces*. To understand the idea of a subtrace, consider the view of a trace as a two-dimensional table, where one dimension is the space of system variables and the other dimension is time. A *subtrace* is a finite window, or a snapshot, of a trace. Thus, just as a movie is a sequence of overlapping images, a trace is a sequence of overlapping subtraces. Restricting ourselves to Boolean variables, each subtrace can be viewed as a binary matrix. Given a set of finite-length traces, and an integer p, the traces can be divided into subtraces of time-length p. The set of all such subtraces constitutes a set of binary matrices. The basis subtraces are simply a set of subtraces that form a basis of the set of subtraces, in that every subtrace can be expressed as a superposition of the basis subtraces.

The form of superposition depends on the type of system being analyzed. In this paper, we focus on digital systems, and more concretely on digital circuits. In this context, one can define superposition as a "linear" combination over the semi-ring with Boolean OR as the additive operator and Boolean AND as the multiplicative operator. The coefficients in the resulting linear combination are either 0 or 1. The problem of computing a basis of a set of subtraces is equivalent to a Boolean matrix factorization problem, in which a Boolean matrix must be decomposed into the product of two other Boolean matrices. If we seek the basis of the smallest size, the problem is equivalent to finding the *ambiguous rank* [9] of the Boolean matrix, which is known to be NP-complete [24].

Given a set of subtraces, several bases are possible. Following Occam's Razor principle, we seek to compute a "simple" basis that generalizes well to unseen traces. More concretely, we seek to find a basis that is minimal in that each subtrace is a linear combination of only a small number of basis subtraces. This yields the *sparse basis problem*. In this paper, we formally define this problem in the context of Boolean matrix factorization and propose a graph-theoretic algorithm to solve the sparse-version of the problem. Such a problem is often referred to as a *sparse coding* problem in the machine learning literature, since it involves encoding a data set with a "code" in a sparse manner using few non-zero coefficients.

We apply the generated basis subtraces to the problem of error localization. In digital circuits, an especially vexing problem today is that of post-silicon debugging, where, given an error trace with potentially only a subset of signals observable and no way to reproduce the trace, one must localize the problem in space (to a small collection of error modules) and time (to a small window within the trace). Similar problems arise in debugging distributed systems. In addition, error localization is very relevant to "pre-silicon" verification as well. Our approach is to attempt to reconstruct windows of an error trace using a basis computed from slicing a set of good traces into subtraces of length p. The hypothesis is that the earliest windows that cannot be reconstructed are likely to indicate the time of the error, and the portions that cannot be reconstructed are likely to indicate the signals (variables) that are the source of the problem. The technique can thus be applied for *simultaneous* error localization and explanation. We apply this technique to representative digital circuits.

To summarize, the main contributions of the paper are:

- We introduce the idea of *basis subtraces* as a formal way of capturing behavior of a design as exhibited by a set of traces;
- We formally define the *sparsity-constrained Boolean matrix factorization problem* and propose a graph-theoretic algorithm to solve it;

- We demonstrate with experimental results that we can mine useful specifications using our sparse coding method, and
- We show that the computed bases can be effective for simultaneous error localization and error explanation, even for transient errors, such as bit flips, that arise not just due to logical errors but also from electrical effects.

Organization. We begin in Sec. 2 with basic terminology and preliminaries. Sec. 3 introduces our approach to finding a sparse basis. In Sec. 4, we show how we can use our approach for performing error localization. Experimental results are presented in Sec. 5. Related work is surveyed in Sec. 6 and we conclude in Sec. 7.

2 Preliminaries

In this section, we introduce the basic notation used in the rest of the paper. Sec. 2.1 introduces notation representing traces of a reactive system as matrices, and Sec. 2.2 connects the matrix representation with a graph representation.

2.1 Traces and Subtraces

We model a reactive system as a transition system (V, Σ_0, δ) where V is a finite set of Boolean variables, Σ_0 is a set of initial states of the system, and δ is the transition relation. In general, V contains input, output and (internal) state variables. A state of the system σ is a Boolean vector comprising valuations to each variable in V. For clarity, we restrict ourselves in this paper to synchronous systems in which transitions occur at the tick of a clock, such as digital circuits, although the ideas can be applied in other settings as well.

Let the state of the system at the ith cycle (step) be denoted by σ_i. A *complete trace* of the system of length l is a sequence of states $\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_{l-1}$ where $\sigma_0 \in \Sigma_0$, and $(\sigma_{i-1}, \sigma_i) \in \delta$ for $1 \leq i < l$. Note however that the full system state and/or inputs might not be observed or recorded during execution. We therefore define a *trace* τ as a sequence of valuations to an observable subset of the variables in V; i.e., $\tau = \sigma'_0, \sigma'_1, \sigma'_2, \ldots, \sigma'_{l-1}$ where $\sigma'_i \subseteq \sigma_i$. A *subtrace* $\tau_{i,j}$ of length j in τ is defined as the segment of τ starting at cycle i and ending at cycle $i + j - 1$, such that $i \geq 0$, $j > 1$ and $i + j \leq l$, i.e. $\tau_{i,j} = \sigma'_i, \sigma'_{i+1}, \ldots, \sigma'_{i+j-1}$. We consider subtraces of length at least 2; i.e., containing at least one transition.

For example, Equation 1 shows a trace τ of length 4 where each state comprises a valuation to two Boolean variables. We depict the trace in matrix form, where the rows correspond to variables and the columns to cycles.

$$\begin{matrix} 1\ 0\ 1\ 1 \\ 1\ 0\ 1\ 1 \end{matrix} \tag{1}$$

The subtrace $\tau_{0,2}$ of τ is

$$\begin{matrix} 1\ 0 \\ 1\ 0 \end{matrix}$$

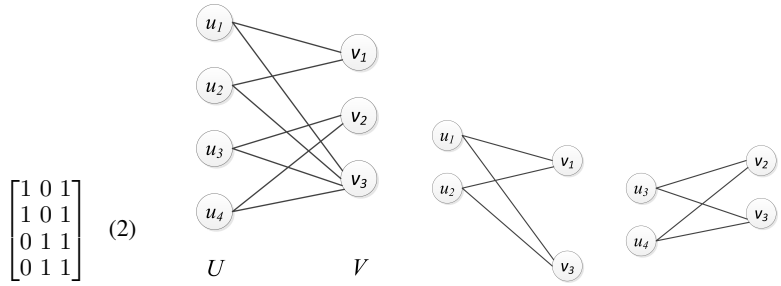
Let $\mathcal{T}_p$ be the set of all subtraces of length p in τ, i.e. $\mathcal{T}_p = \{\tau_{i,p} | 0 \leq i \leq l - p\}$. For any $\tau_{i,p} \in \mathcal{T}_p$, we can view it as a Boolean matrix of dimension $|V| \times p$. We can also represent it using a vector $v_i^p \in \mathbb{B}^{|V| \times p}$ by stacking the columns in $\tau_{i,p}$ (i.e., using a column-major representation). For example, v_0^2 as shown below represents the subtrace $\tau_{0,2}$.

$$v_0^2 = [1\ 1\ 0\ 0]^T$$

For brevity, we use v_i for v_i^p when the length of each subtrace p is obvious from the context. Hence, we can represent $\mathcal{T}_p$ as a Boolean matrix with $|V| \times p$ rows and $l-p+1$ columns. For example, we can represent all the subtraces of length 2 for the trace in Equation 1 as the matrix in Equation 2 in Fig. 1(a).

2.2 Boolean Matrices and Bipartite Graphs

A Boolean matrix can be viewed as an adjacency matrix for a *bipartite graph* (*bigraph*, for short). Recall that a bipartite graph $G = \langle U, V, E \rangle$ is a graph with two disjoint non-empty sets of vertices U and V and such that every edge in $E \subseteq U \times V$ connects one vertex in U and one in V. For a Boolean matrix $M \in \mathbb{B}^{k_1 \times k_2}$, denote $M_{i,j}$ as the entry in the i^{th} row and j^{th} column of M. Then, M can be represented by a bigraph G_M with $U = \{u_1, u_2, \ldots, u_{k_1}\}$ and $V = \{v_1, v_2, \ldots, v_{k_2}\}$, such that there is an edge connecting $u_i \in U$ and $v_j \in V$ if and only if $M_{i,j} = 1$. For example, the matrix X in Equation 2 (Fig. 1(a)) can be represented by the bigraph G_X in shown in Fig. 1(b).



(a) Matrix form (b) Bipartite graph (c) Biclique edge cover

Fig. 1. Subtraces in matrix and bigraph form, and corresponding biclique edge cover

A *biclique* is a complete bipartite graph; i.e., a bipartite graph $G' = \langle U', V', E' \rangle$ where $E' = U' \times V'$. Given a bigraph G, a *maximal edge biclique* of G is a biclique $B_1 = \langle U_1 \subseteq U, V_1 \subseteq V, E_1 = U_1 \times V_1 \rangle$ if it is not contained in another biclique of G, that is, there does not exist another biclique $B_2 = \langle U_2 \subseteq U, V_2 \subseteq V, E_2 = U_2 \times V_2 \rangle$ and either $U_1 \subset U_2$ or $V_1 \subset V_2$. In the rest of the paper, we use the pair of vertices (U_1, V_1) to denote the maximal edge biclique B_1. For a set of bicliques Cov and a bigraph G, denote E_{Cov} as the set of edges in G covered by Cov, i.e. $\forall\, e \in E_{Cov}, \exists\, G' = \langle U' \subseteq U, V' \subseteq V, E' \rangle \in Cov,$ s.t. $e \in E'$. Cov is a *biclique edge cover* of G if and only if all the edges E in G are covered by the set, i.e. $E_{Cov} = E$. Abusing notation a little, we use E_v to denote the set of edges connected to vertex v. The smallest number of bicliques needed is called the *bipartite dimension* of G. For example, a biclique cover for the bigrah in Figure 1(b) is shown in Figure 1(c).

The view of Boolean matrices as bigraphs is relevant for decomposing a set of traces into a set of basis subtraces. The following problem is important in this context.

Definition 1. *Consider a Boolean matrix $X \in \mathbb{B}^{m \times n}$, the* Boolean matrix factorization problem *is to find k and Boolean matrices $B \in \mathbb{B}^{m \times k}$ and $S \in \mathbb{B}^{k \times n}$ such that*

$$X = B \circ S \tag{3}$$

That is, X is decomposed into a Boolean combination (denoted by the operator $\circ$) of two other Boolean matrices, in which scalar multiplication is the Boolean AND operator $\wedge$, and scalar addition ("+") is the Boolean OR operator $\vee$. In other words, we perform matrix/vector operations over the Boolean semi-ring with $\wedge$ as the multiplicative operator and $\vee$ as the additive operator. For example, the matrix in Equation 2 (Fig. 1(a)) can be factorized in the following way.

$$\begin{bmatrix}1&0&1\\1&0&1\\0&1&1\\0&1&1\end{bmatrix} = \begin{bmatrix}1&0\\1&0\\0&1\\0&1\end{bmatrix} \circ \begin{bmatrix}1&0&1\\0&1&1\end{bmatrix}$$

We use $M_{.,i}$ to denote the i^{th} column vector of a matrix M, and $M_{i,.}$ to denote the i^{th} row vector of M. Thus, the columns of matrix X are $X_{.,1}, X_{.,2}, \ldots, X_{.,n}$. We will refer to X as the *data matrix* since it represents the traces which are the input data. We call the matrix B the *basis matrix* because each $B_{.,i}$ can be viewed as some basis vector in $\mathbb{B}^m$. We call the matrix S the *coefficient matrix*. Each $S_{.,i}$ is a Boolean vector in which a 1 in the j^{th} entry indicates that the j^{th} basis vector is used in the decomposition and 0 otherwise.

We can also rewrite the factorization in the following way as a Boolean sum of the matrices formed by taking the tensor (outer) product of the i^{th} column in B and the i^{th} row in S.

$$\begin{bmatrix}1&0&1\\1&0&1\\0&1&1\\0&1&1\end{bmatrix} = \begin{bmatrix}1&0&1\\1&0&1\\0&0&0\\0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0\\0&0&0\\0&1&1\\0&1&1\end{bmatrix}$$

Notice that the two matrices on the right hand side represent the bicliques in Fig. 1(c).

Remark 1 *Clearly, a solution always exists for the problem in Definition 1. This is because one can always pick $k = n$ such that $B = I(B = X)$ and $S = X(S = I)$ (where I is the identity matrix). However, this is not particularly revealing in terms of the behaviors which each $X_{.,i}$ is composed of. One alternative is to minimize k. The smallest k for which such a decomposition exists is called the ambiguous rank [9] of the Boolean matrix X. It is also equal to the bipartite dimension of the bigraph G_X corresponding to matrix X. The problem of finding a Boolean factorization of X with the smallest k is equivalent to finding a biclique edge cover of G_X with the minimum number of bicliques. Both problems are NP-hard [24]. On the other hand, one can choose to find an overcomplete basis ($k > n$) such that each $X_{.,i}$ can be expressed as a Boolean sum of only a few basis vectors. We discuss this formulation in detail in Section 3.*

3 Specification Mining via Sparse Coding

In this section, we describe how specifications are mined via sparse Boolean matrix factorization. The specifications we mine, *basis subtraces*, can be viewed as temporal patterns over a finite time window.

3.1 Formulation as Sparse Coding Problem

The notion of sparsity is borrowed from the wealth of literature in machine learning such as sparse coding [15] and sparse principal component analysis (PCA) [30]. *The key insight is that a sparsity constraint often generates a better interpretation of the data in terms of the underlying concepts.* In the setting of mining specifications from a trace, we argue that each subtrace of a trace can be viewed as a superposition of patterns, and a potential specification is a pattern that is commonly shared by multiple subtraces. These patterns are the so-called *basis subtraces*.

We present the *sparse Boolean matrix factorization problem* for computing basis subtraces below. A few different options are presented for formulating the problem and we pick one with a notion of sparsity that seems well-suited to our context.

Definition 2. *Given $X \in \mathbb{B}^{m \times n}$ and a positive integer C, the* sparsity-constrained Boolean matrix factorization problem *is to find k, $B \in \mathbb{B}^{m \times k}$, and $S \in \mathbb{B}^{k \times n}$ such that*

$$\begin{aligned} X &= B \circ S \\ \text{and} \quad \|S_{\cdot,i}\|_1 &\leq C, \, \forall_i \end{aligned} \tag{4}$$

Let us reflect on the above problem formulation. The constraint $X = B \circ S$ imposes the requirement that the input data (subtraces) represented by X must be reconstructed as a superposition of the subtraces represented by B, with S encoding the coefficients in the superposition. The second constraint $\|S_{\cdot,i}\|_1 \leq C, \forall_i$ encodes the sparsity constraint, which ensures that each subtrace in X is a sparse superposition of the subtraces in B.

More precisely, the definition above imposes a constraint on the number of 1s per column of S. Similar to the Boolean matrix factorization problem in Definition 1, a trivial solution is to set $B = X$ and $S = I$ (and $k = n$). However, this solution does not produce any sharing of patterns amongst the different subtraces and hence is useless for specification mining. The optimization objective is thus the following, which maximizes the number of 1s in S.

$$\text{maximize} \sum_i \sum_j S_{i,j} \tag{5}$$

We describe how we solve this problem in Section 3.2.

One might also consider defining sparsity in a somewhat different manner. Instead of imposing a L_1-norm constraint on the columns of the coefficient matrix S, we can seek B and S such that the total sparsity is minimized.

Definition 3. *Given $X \in \mathbb{B}^{m \times n}$ and a positive integer k, the* sparsity-optimized Boolean matrix factorization problem *is the following optimization problem.*

$$\begin{aligned} \underset{B,S}{\text{minimize}} \quad & \sum_i^n \|S_{\cdot,i}\|_1 \\ \text{subject to} \quad & X = B \circ S \end{aligned} \tag{6}$$

The main issue with this problem definition is that k is fixed; in other words, one has to "guess" a suitable k for which B and S can be computed. While modifications of this problem that restrict or minimize k could potentially be useful, we leave the investigation of these variants of 6 to future work.

3.2 Solving the Sparse Coding Problem

In this section, we describe an algorithm that solves the *sparsity-constrained Boolean matrix factorization problem*, as formalized in Equations 4 and 5. Our solution is guaranteed to satisfy the sparsity constraint and tries to maximize the objective in Equation 5. The algorithm exploits the connection between the matrix factorization problem and the biclique edge cover problem described in Sec. 2. Specifically, it is based on growing a biclique edge cover Cov for the bigraph $G_X = \langle U, V, E \rangle$ corresponding to matrix X. At each step, a maximal edge biclique that covers some number of previously uncovered edges is added to Cov until Cov covers all the edges. The sparsity constraint is then a constraint on the number of maximal bicliques that can be used to cover the edges that connect each vertex in V. (Recall that each vertex in V corresponds to a column $S_{\cdot,i}$ of S.)

Notice that this algorithm relies on a way to generate maximal edge bliciques of a bigraph. Computing these bicliques is not easy: for instance, the closely-related problem of finding a maximum (not maximal) edge biclique in a bigraph is NP-complete [23]. Additionally, the number of maximal bicliques in a bigraph can be exponential in the number of vertices [11].

However, there exist enumeration algorithms that are polynomial in the combined input and output size, such as the Consensus algorithm in [1]. In addition, this algorithm runs in incremental polynomial time.

Algorithm 1 solves the sparsity-constrained Boolean matrix factorization problem by building upon some key concepts in the Consensus algorithm and adapting them for our problem context. These concepts are described below.

- **Consensus:** For two bicliques $B_1 = (U_1, V_1)$ and $B_2 = (U_2, V_2)$, the consensus of B_1 and B_2 is $B_3 = (U_3, V_3)$ where $U_3 = U_1 \cap U_2$ and $V_3 = V_1 \cup V_2$.
- **Extend to a maximal biclique:** For a consensus biclique $B_1 = (U_1, V_1)$, we can extend it to a maximal biclique $B_2 = (U_2, V_2)$ where $U_2 = U_1$ and $V_2 = \{v \mid \forall\, u \in U_1, (u, v) \in E\}$
  (V_2 is the set of vertices in V that are connected to every vertex in U_1).
- **v-rooted star biclique:** A v-rooted star biclique is the biclique formed by the node $v \in V$ and all the nodes connected to v (and the edges), i.e. $(\{u \mid (u, v) \in E\}, \{v\})$

The main idea of Algorithm 1 is the following. We try to cover the edges in the bigraph with as many maximal bicliques as possible, until we are about to violate the sparsity constraint at some vertex $v \in V$. In that case, we cover the remaining edges of v with the v-rooted star biclique. If there is still some $v \in V$ with uncovered edges at the end of the iteration, then we just cover it with the v-rooted star biclique as well. The final cover will be the union of the set of maximal bicliques added in the consensus steps $Cov_1 \setminus Cov_0$ with the set of star bicliques Cov_2.

4 Application to Error Localization

The key idea in our approach is to localize errors by attempting to reconstruct the error trace from basis subtraces generated from correct traces. Our hypothesis is that the earliest section (subtrace) of the error trace that cannot be reconstructed contains the likely cause of the error. Our localization algorithm is presented in this section, along with some theoretical guarantees. We begin with the problem definition.

Algorithm 1 Sparsity-constrained cover

Input: the set of v-rooted star bicliques Cov_0 and sparsity constraint C.
Initialize: $Cov_1 := Cov_0$, $Cov_2 := \emptyset$, $\alpha_v := C,\ \forall\, v \in V$, and $V_{cov} := \emptyset$.
repeat
  Pick a new pair of bicliques $B_1 = (U_1, V_1)$ from Cov_1 and $B_2 = (U_2, V_2)$ from Cov_0, form the consensus B_3.
  Extend B_3 to a maximal biclique $B_4 = (U_4, V_4)$.
  if $(B_4 \notin Cov_1) \wedge (V_4 \cap V_{cov} = \emptyset)$ **then**
    Add B_4 to Cov_1.
    for $v \in V_4 \setminus V_{cov}$ **do**
      $\alpha_v := \alpha_v - 1$
      if $\alpha_v = 1$ **then** Add the v-rooted star biclique to Cov_2 and add v to V_{cov} **end if**
    end for
  end if
until $E_{(Cov_1 \setminus Cov_0) \cup Cov_2} = E$ or cannot find a new pair of bicliques B_1 and B_2
for $v \in V \setminus V_{cov}$ **do**
  Add the v-rooted star biclique to Cov_2.
end for
Output: the sparsity-constrained cover $(Cov_1 \setminus Cov_0) \cup Cov_2$

4.1 Problem Definition

Consider the problem of localizing an error given a set of correct traces and a single error trace. Our goal is to identify a small interval of the timeline at which the error occurred. What makes the problem especially challenging is that the input sequence that generated the error trace is either unknown (or only partially known) or it is extremely slow to re-simulate the input sequence (if known) on the correct design (also sometimes referred to as a "golden model"). This means that a simple anomaly detection technique which checks the first divergence of the error trace and the correct trace obtained by simulating the golden model on the same input sequence does not work. One has to use the set of correct traces to help localize the bug in the error trace. This setting is especially applicable to post-silicon debugging where the bugs are often difficult to diagnose due to limited observability, limited reproducibility and susceptibility to environmental variations.

More formally, the error localization problem we address in this section can be defined as follows.

Definition 4. *Given an error trace of length l and an integer p, partition the trace into non-overlapping subtraces each of length p (w.l.o.g. assume l is an integer multiple of p; otherwise, the last subtrace can be treated specially).*

Then, the error localization problem *is to identify the subtrace containing the first point of deviation of the error trace from the correct trace on the same input sequence.*

One might note that the problem we define is not the only form of error localization that is desirable. For instance, one might also want to narrow down the fault to the signals/variables that were incorrectly updated.

Also, there might be more than one source of an error, in which case one might want to identify all of the sources.

While these goals are important, we contend that our algorithm to address the problem defined above can also be used to achieve these additional objectives. For example,

the error explanation technique we present below can be used to identify which variables were incorrectly updated and how. Similarly, one can apply our reconstruction-based localization algorithm iteratively to identify multiple subtraces that cannot be reconstructed from the basis subtraces, and could potentially be used to identify multiple causes of an error.

4.2 Localization by Reconstruction

As described above, the key hypothesis underlying our approach is that the earliest section (subtrace) of the error trace that cannot be reconstructed contains the likely cause of the error.

Our error localization algorithm operates in the following steps:

1. Given a set of correct traces $\mathcal{T}$, first obtain the set of all *unique* subtraces of length p in $\mathcal{T}$. Denote this set by $\mathcal{T}_p$. Using the approach described in Section 2, convert the set $\mathcal{T}_p$ to a data matrix X.
2. Solve the *sparsity-constrained Boolean matrix factorization problem* for X for a given constant C.
3. Given an error trace τ', partition it into an ordered set of q subtraces of length p. Denote this set by $\mathcal{T}'_p$. The elements in $\mathcal{T}'_p$ are ordered by their positions in τ'. Convert $\mathcal{T}'_p$ to a data matrix X'.
4. Starting from $X'_{\cdot,0}$, try to reconstruct $X'_{\cdot,i}$ using the basis computed above with the same sparsity constraint C. Return i as the location of the bug if the reconstruction fails. In case all reconstructions succeed, return $\perp$ indicating inability to localize the error.

Algorithm 2 describes the above approach in more detail using pseudo-code. It uses the following subroutines:

- **dataMatrix** is the procedure that converts a set of subtraces to the corresponding data matrix described in Section 2.
- **sparseBasis** solves the *sparsity-constrained Boolean matrix factorization problem* using the graph-theoretic algorithm presented in Section 3 for X with a given C, and returns the computed basis B.
- **reconstructTrace** solves the following minimization problem.

$$\begin{aligned} \underset{S_i}{\text{minimize}} \quad & \|X'_{\cdot,i} \oplus (B \circ S_{\cdot,i})\|_1 \\ \text{subject to} \quad & \|S_{\cdot,i}\|_1 \leq C \end{aligned} \tag{7}$$

  where $\oplus$ is the bit-wise Boolean XOR operator, and is interpreted to apply entry-wise on matrices.

  Notice that for fixed C, this problem is *fixed-parameter tractable* because we can use a brute-force algorithm that enumerates all the $\sum_{1 \leq i \leq C} \binom{k}{i}$ possible $S_{\cdot,i}$. It can also be solved using a pseudo-Boolean optimization formulation, where the Boolean variables in the optimization problem are the entries in $S_{\cdot,i}$.

Error Explanation. Denote $S^*_{\cdot,i}$ as the optimal solution to the minimization problem in Equation 7. If the minimum value is non-zero, then $E = X'_{\cdot,i} \oplus (B \circ S^*_{\cdot,i})$ is the minimum difference between the error subtrace $X'_{\cdot,i}$ and the reconstructed subtrace $B \circ S^*_{\cdot,i}$. Notice that E is also a subtrace, and can be interpreted as a finite sequence of assignments to system variables. In our experience, E is a pattern that explains the error; we expand further on this point using our experiments in Sec. 5.

Algorithm 2 Error localization in time

Input: Set of subtraces $\mathcal{T}_p$ from set of correct traces $\mathcal{T}$, $\mathcal{T}'_p$ from error trace τ'
Input: Constant $C > 0$
$X =$ **dataMatrix**$(\mathcal{T}_p)$; $X' =$ **dataMatrix**$(\mathcal{T}'_p)$; $B =$ **sparseBasis**(X, C)
for $i := 0 \rightarrow q - 1$ **do**
$E =$ **reconstructTrace**$(X'_{,i}, B, C)$
if $E \neq \mathbf{0}$ **then** **return** i **end if**
end for
return $\perp$

4.3 Theoretical Guarantees

We now give conditions under which our error localization approach is *sound*. By sound, we mean that when our algorithm reports a subtrace as the cause of an error, it is really an erroneous subtrace that deviates from correct behavior.

Since our approach mines specifications from traces, its effectiveness fundamentally depends on the quality of those traces. Specifically, our soundness guarantee relies on the set of traces $\mathcal{T}$ satisfying the following *coverage metrics* defined over the transition system (V, Σ_0, δ) of the golden model:

1. *Initial State Coverage:* For every initial state $\sigma_0 \in \Sigma_0$, there exists some trace in $\mathcal{T}$ in which σ_0 is the initial state.
2. *Transition Coverage:* For every transition $(\sigma, \sigma') \in \delta$, there exists some trace in $\mathcal{T}$ in which the transition (σ, σ') occurs.

While full transition coverage can be difficult to achieve for large designs, there is significant work in the simulation-driven hardware verification community on achieving a high degree of transition coverage [25]. If achieving transition coverage is challenging for a design, one could consider slicing the traces based on smaller module boundaries and computing tests that ensure full transition coverage within modules, at the potential cost of missing cross-module patterns.

Our soundness theorem relates test coverage with effectiveness of error localization.

Theorem 1. *Given a transition system Z for the golden model and a set of finite-length traces $\mathcal{T}$ of Z satisfying initial state and transition coverage, if Algorithm 2 is invoked on $\mathcal{T}$ and an arbitrary error trace τ', then Algorithm 2 is sound; viz., if it reports a subtrace of τ' as an error location, that subtrace cannot be exhibited by Z.*

Proof. (sketch) The proof proceeds by contradiction. Suppose Algorithm 2 reports a subtrace of τ' as the location of the error. Recall that a subtrace must be of length at least 2. Thus, if we compute basis subtraces of length 2, any transition of the golden model Z can be expressed as a superposition of these basis subtraces and hence reconstructed from the basis subtraces B, since $\mathcal{T}$ contains all transitions of Z. A subtrace reported as an error location, in contrast, is one that cannot be expressed as a superposition of the basis subtraces and hence **reconstructTrace** will report that it cannot be reconstructed. Thus, any subtrace reported as an error location by Algorithm 2 cannot be a valid transition of the golden model Z. □

We also note that, in theory, it is possible for Algorithm 2 to miss reporting a subtrace that is an error location, if that subtrace is expressible as a superposition of basis subtraces. However, experiments indicate that it is usually accurate in pinpointing the location of the error. Details of our experiments are provided in Sec. 5.

5 Experimental Results

In this section, we evaluate our sparse coding approach to generate specifications and localize errors based on the following criteria.

(1) Are the computed "basis subtraces" meaningful? That is, do they correspond to some interesting specifications of the test circuit?
(2) Do the "basis subtraces" capture sufficient underlying structure of a trace? That is, can they be used to reconstruct traces that are generated from unseen input sequences?
(3) How accurately can we localize an error in an unseen trace (generated by unseen input sequences)?
(4) How good are the error explanations?

5.1 Arbiter

We first use a 2-port arbiter as an illustrative example to evaluate our approach. The 2-port arbiter is a circuit that takes two Boolean inputs corresponding to two potentially competing requests, and produces two Boolean outputs corresponding to the two grants. It implements a round-robin scheme such that it will give priority to the port at which a request has not been most recently granted. Let r_0, r_1 denote the input requests and g_0, g_1 denote the corresponding output grants. If a request r_i is granted, g_i goes high in the same cycle. Figure 2 shows part of a trace of the arbiter over the request and grant signals. The input requests were randomly generated and the trace was 100 cycles long.

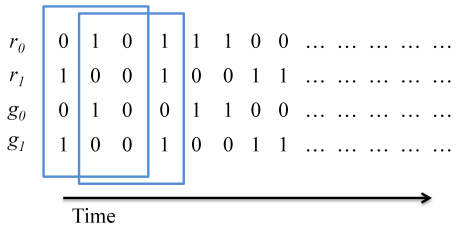


Fig. 2. A normal trace of a 2-port round-robin arbiter

We used a sliding window of length 3 to collect a set of subtraces. We then applied our sparse coding algorithm described in Section 3.2 to extract a set of "basis subtraces". We set the sparsity constraint to 4 for this experiment, which is only one third of the total number of entries in a subtrace. We now evaluate our approach with respect to the four criteria stated at the start of this section:

(1) Figure 3 shows some of the basis subtraces computed. We can observe that basis (a) and (b) correspond to the correct behavior of the arbiter granting a request at the same cycle when there is no competing request. Basis (c) shows that when there are two competing requests at the same cycle, the arbiter first grants one of the requests and the ungranted request will stay asserted the next cycle and then gets granted.
(2) We further simulated the arbiter with random inputs another 100 times each for 100 cycles. For each of these traces, we also use a sliding window to partition them into subtraces of length 3. Using the basis computed from the trace depicted in Figure 2, we tried to reconstruct these subtraces and succeeded in every attempt. This was

0 0 0	0 1 0	0 1 1
1 0 0	0 0 0	0 1 0
0 0 0	0 1 0	0 0 1
1 0 0	0 0 0	0 1 0
(a)	(b)	(c)

Fig. 3. Three basis subtraces computed via sparse coding

because all the sub-behaviors were fully covered in the trace from which the bases were computed, even though unseen subtraces exist in the new traces.

(3) For each of the 100 traces in (2), we randomly injected a single bit error (flipping its value) at a random cycle to one of the four signals in the trace. Our task was to test if we could localize the error to a subtrace of length 3 that contained it.
The following example illustrates one of the experiments. Figure 4(a) shows a snapshot of the trace.

	95	96	97	98	99
r_0	0	0	0	1	0
r_1	0	0	1	0	0
g_0	0	0	0	1	0
g_1	0	0	**0**	0	0

(a) Bit flip at r_1 at cycle 97

	96	97	98
r_0	0	0	1
r_1	0	1	0
g_0	0	0	1
g_1	0	0	0

(b) Error subtrace as identified

	96	97	98
r_0	0	0	0
r_1	0	1	0
g_0	0	0	0
g_1	0	0	0

(c) Error explanation subtrace

	96	97	98
r_0	0	0	0
r_1	0	0	0
g_0	0	0	0
g_1	0	1	0

(d) Alternative error explanation subtrace

Fig. 4. Error trace and explanation subtrace

Using the approach described in Algorithm 2, the subtrace containing the error was correctly identified. Among the 100 traces, we successfully identified the window at which the error was injected for 84 of them. Figure 4(b) shows the error subtrace.
Following Equation 7, Figure 4(c) shows the (differential) subtrace $X_i' \oplus (B \circ S_i)$ that minimizes $|X_i' \oplus (B \circ S_i)|_1$ and serves as an error explanation.
Clearly, this subtrace reveals the injected error. While no fault model is assumed, this approach still pinpoints the bug behaviorally – a grant was not produced at g_1 at cycle 97 even when the corresponding request was made at r_1. Note that multiple error explanations (solutions to the minimization problem in Equation 7) can exist. Figure 4(d) shows an alternative error explanation subtrace for this example where g_1 was asserted but r_1 was not asserted at cycle 97.

(4) In Section 4.2, we argue that the *minimum difference* between an error subtrace and any possible reconstructed subtrace using the computed basis can serve as an explanation for the error. In the 84 traces for which the error was correctly localized, the injected bit error was also uncovered by solving the optimization problem in Equation 7.

5.2 Chip Multiprocessor (CMP) Router

Our second, larger case study is a router for on-chip networks. The main goal of this case study was to explore how the technique scales to a larger design, and how effective it is for error localization.

Fig. 5 illustrates the high-level design of the router, as a composition of four high-level modules. The input controller comprises a set of FIFOs buffering incoming flits

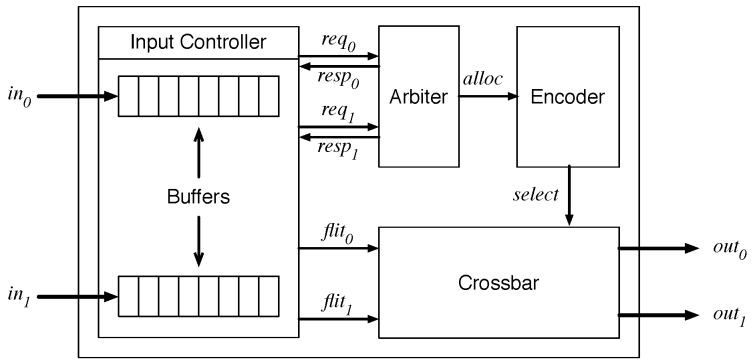


Fig. 5. CMP Router comprising four high-level modules

and interacting with the arbiter. When the arbiter grants access to a particular output port, a signal is sent to the input controller to release the flits from the buffers, and at the same time, an allocation signal is sent to the encoder which in turn configures the crossbar to route the flits to the appropriate output port.

The router was simulated with two flit generating modules that each issued random data packets (each consists of a head, some body and a tail flit) to the respective input ports of the router. We observed 14 Boolean control signals in the router and a trace was generated for these 14 signals with a simulation length of 1000 cycles. We used a subtrace width of 2 cycles and obtained 93 distinct subtraces each with 14 signals over 2 cycles. A basis was computed from these 93 distinct subtraces subject to a sparsity constraint of 52 (see explanation for the choice of this number at the end of this section). It took 0.243 seconds to obtain this basis which contained 189 basis subtraces.

The router was simulated 100 times with different inputs. We used the first simultion trace to obtain the basis as described in the previous paragraph and the rest 99 traces for error localization. For each of these 99 traces, a single bit flip was injected to a random signal at a random cycle. The goal of experiment is to localize this bit error to a subtrace of 2 cycles (among the 999 subtraces for each trace) in which the error was introduced.

Following the localization approach described in Section 4.2 of the paper, 55 out of 99 of the errors were correctly localized. The remaining 44 errors were not localized (all the subtraces including error subtrace were reconstructed using the computed basis). The overall accuracy of the error localization procedure in this experiment was 55.6%.

Why is this error localization approach useful? Imagine you are given a good trace (or a collection of good traces) and then an error trace (that cannot be reproduced), and you are asked to localize the error without knowing very much about the underlying system that generates these traces. (This situation arises when dealing with legacy systems, for example.) Here are two plausible alternative options to our sparse coding approach and the corresponding results:

(a) Hash all the distinct subtraces of 2 cycles in length in the good trace. For each of the subtraces of the same dimension in the bad trace, check if it is contained in the hash, and report an error if it is not contained. For the same traces used above, an error was reported for each of the 99 traces even before any bit flip was injected.
(b) Use a basis that spans the entire space of subtraces of 2 cycles, e.g. 14×2 subtraces where each contains only a single 1 in its entries and is orthogonal to the others. However, it is obvious that we cannot localize any error using this basis since it spans all possible subtraces.

Our method can be viewed as something in between (a) and (b). It finds a subspace that not only contains all the good subtraces but also generalizes well to unseen good subtraces from the basis. The generalization is a sparse composition of some key patterns in the good subtraces. An error is reported if a subtrace lies outside this subspace. The number 52 for the sparsity constraint was determined as a result of the minimization of sparsity such that the computed basis was just sufficient to reconstruct all the other 99 traces before error injection. This limits the size of the subspace spanned by the basis and hence increases the ability to detect an error.

6 Related Work

We survey related work along three dimensions: Boolean matrix factorization, mining specifications from traces, and error localization techniques.

6.1 Boolean Matrix Factorization

Matrix factorization or factor analysis methods are prevalent in the data mining community, with a common goal to discover the latent structures in the input data. While most of these methods are focusing on real-valued matrices, there have been several works recently that target Boolean matrices, for applications such as role mining [26]. Miettinen et al. [20] introduced the discrete basis problem (DBP). DBP is similar to our definition of the Boolean matrix factorization problem in which k is fixed and the objective is to minimize the reconstruction error. They showed that DBP is NP-hard and gave a simple greedy algorithm for solving it. In terms of sparse decomposition, Miettinen [19] showed the existence of sparse factor matrices for a sparse data matrix. Our paper describes a different notion of sparsity – we seek to express each data vector as a combination of only a few basis vectors, which can be dense themselves.

6.2 Specification Mining

Approaches to mine specifications can be largely categorized into static and dynamic methods. We restrict ourselves here to the dynamic methods that mine specifications from traces. Daikon [8] is one of the earliest tools that mine single-state invariants or pre-/post-conditions in programs. In contrast, we focus on mining (temporal) properties over a finite window for reactive (hardware) designs. Some existing tools produce temporal properties in the form of automata. Automata-based techniques generally fall into two categories. The first class of methods learn a single complex specification (usually as a finite automaton) over a specific alphabet, and then extract simpler properties from it. For instance, Ammons et al. [2] first produce a probabilistic automaton that accepts the trace and then extract from it likely properties. However, learning a single finite state machine from traces is NP-hard [12]. To achieve better scalability, an alternative is to first learn multiple small specifications and then post-process them to form more

complex state machines. Engler et al. [7] first introduce the idea of mining simple alternating patterns. Several subsequent efforts [27, 28, 10] built upon this work. In previous work, we proposed a specification mining approach similar to Javert that focuses on patterns relevant for digital circuits [16] and showed how this can be applied to error localization. However, such approaches are limited by the set of patterns. The present work seeks to remove this limitation by inferring design-specific patterns in the form of basis subtraces.

6.3 Error Localization

The problem of error localization and explanation has been much studied in literature in several communities: software testing, model checking, and electronic design automation. In model checking, Groce et al. [13] present an approach based on distance metrics which, given a counterexample (error trace), finds a correct trace as "close" as possible to the error trace according to the distance metrics. Ball et al. [3] present an approach to localizing errors in sequential programs. They use a model checker as a subroutine, with the core idea to identify transitions of an error trace that are not in any correct trace of the program, and use this for error localization. Both of these approaches operate on error traces generated by model checking, and thus have full observability of the inputs and state variables. In contrast, in our context, the trace includes only-partially observed state and is not reproducible.

In the software testing community, researchers have attempted to use predicates and mined specifications to localize errors [17, 6]; however, these rely on human insight in choosing a good set of predicates/templates. In contrast, our approach automatically derives specifications in the form of basis subtraces, which can be seen as temporal properties over a finite window. Program spectra [14], which include computing profiles of program behavior such as summaries of the branches or paths traversed, have also been proposed as ways to separate good traces from error traces; however, these techniques are of limited use for digital circuits since they rely on the path structure of sequential programs and give no guarantees on soundness.

In the area of post-silicon debugging (see [21] for a recent survey), the problem of error localization has received wide attention. The IFRA approach [22], which is largely specialized for processor cores, is based on adding on-chip recorders to a design to collect "instruction footprints" which are analyzed offline with some input from human experts. Li et al. [16] have proposed the use of mined specifications to perform error localization; however, this approach relies on human insight in supplying the right templates to mine temporal logic specifications. Zhu et al. [29] propose a SAT-based technique for post-silicon fault localization, where *backbones* are used to propagate information across sliding windows of an error trace. This additional information helps make the approach more scalable and addresses the problem of limited observability. Backspace [5] addresses the problem of reproducibility by attempting to reconstruct one or more "likely" error traces by performing backwards reachability guided by recorded signatures of system state; such a system is complementary to the techniques proposed herein for error localization.

7 Conclusion and Future Work

In this paper, we have presented *basis subtraces*, a new formalism to capture system behavior from simulation or execution traces. We showed how to compute a *sparse* basis from a set of traces using a graph-based algorithm. We further demonstrated that the generated basis subtraces can be effectively used for error localization and explanation.

In terms of future work, we envisage two broad directions: improving scalability and applying the ideas to other domains. Since the Boolean matrix factorization problem and its sparse variants can be computationally expensive to solve, the scalability of the approach must be improved. In this context, it would be interesting to use slightly different definitions of a basis (for example, using the field of rationals rather than the semi-ring we consider) so that the problem of computing a sparse basis is polynomial-time solvable. Moreover, the ideas introduced in this paper can be extended beyond digital circuits to software, distributed systems, analog/mixed-signal circuits, and other domains, providing many interesting directions for future work.

Acknowledgement

The authors acknowledge the support of the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. This work was also supported in part by an Alfred P. Sloan Research Fellowship and a Hellman Family Faculty Fund Award.

References

1. G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Appl. Math.*, 145:11–21, December 2004.
2. G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *POPL*, pages 4–16, 2002.
3. T. Ball, M. Naik, and S. K. Rajamani. From symptom to cause: localizing errors in counterexample traces. In *POPL*, pages 97–105, 2003.
4. J. R. Buechi. On a Decision Method in Restricted Second-Order Arithmetic. In *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
5. F. M. de Paula, M. Gort, A. J. Hu, S. J. E. Wilton, and J. Yang. Backspace: Formal analysis for post-silicon debug. In *FMCAD*, pages 1–10, 2008.
6. N. Dodoo, L. Lin, and M. D. Ernst. Selecting, refining, and evaluating predicates for program analysis. Technical Report MIT-LCS-TR-914, MIT Laboratory for Computer Science, 2003.
7. Engler, D. et al. Bugs as deviant behavior: a general approach to inferring errors in systems code. In *SOSP*, pages 57–72, 2001.
8. Ernst, M. et al. The daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.*, 69(1-3):35–45, 2007.
9. V. Froidure. *Rangs des relations binaires, semigrollpes de relations non ambigues*. PhD thesis, June 1995.
10. M. Gabel and Z. Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In *FSE*, pages 339–349, 2008.
11. S. Gaspers, D. Kratsch, and M. Liedloff. On independent sets and bicliques in graphs. In *Graph-Theoretic Concepts in Computer Science*, volume 5344 of *LNCS*, pages 171–182. 2008.
12. E. M. Gold. Complexity of automatic identification from given data. 37:302–320, 1978.
13. A. Groce, S. Chaki, D. Kroening, and O. Strichman. Error explanation with distance metrics. *Software Tools for Technology Transfer (STTT)*, 8(3):229–247, 2006.
14. M. J. Harrold, G. Rothermel, K. Sayre, R. Wu, and L. Yi. An empirical investigation of the relationship between spectra differences and regression faults. *Softw. Test., Verif. Reliab.*, 10(3):171–194, 2000.
15. H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *In NIPS*, pages 801–808. NIPS, 2007.
16. W. Li, A. Forin, and S. A. Seshia. Scalable specification mining for verification and diagnosis. In *Design Automation Conference (DAC)*, pages 755–760, June 2010.
17. B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug isolation via remote program sampling. In *PLDI*, pages 141–154, 2003.

18. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
19. P. Miettinen. Sparse boolean matrix factorizations. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 935–940, Washington, DC, USA, 2010. IEEE Computer Society.
20. P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. In *Proceedings of the 10th European conference on Principle and Practice of Knowledge Discovery in Databases*, PKDD'06, pages 335–346, Berlin, Heidelberg, 2006. Springer-Verlag.
21. S. Mitra, S. A. Seshia, and N. Nicolici. Post-silicon validation: Opportunities, challenges and recent advances. In *Proceedings of the Design Automation Conference (DAC)*, pages 12–17, June 2010.
22. S. B. Park, A. Bracy, H. Wang, and S. Mitra. Blog: Post-silicon bug localization in processors using bug localization graphs. In *DAC*, 2010.
23. R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.
24. D. J. Siewert. *Biclique covers and partitions of bipartite graphs and digraphs and related matrix ranks of 0,1 matrices*. PhD thesis, 2000.
25. S. Tasiran and K. Keutzer. Coverage metrics for functional validation of hardware designs. *IEEE Design & Test of Computers*, 18(4):36–45, 2001.
26. J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *In Symposium on Access Control Models and Technologies (SACMAT*, pages 175–184, 2007.
27. W. Weimer and G. C. Necula. Mining temporal specifications for error detection. In *TACAS*, pages 461–476, 2005.
28. Yang, J. et al. Perracotta: mining temporal api rules from imperfect traces. In *ICSE*, pages 282–291, 2006.
29. C. S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In *FMCAD*, 2011.
30. H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15:2006, 2004.