Technical Companion to

A Polynomial-Time Algorithm for Solving the Minimal Observability Problem in Conjunctive Boolean Networks

A Self-Contained Guide for CS/EE/Math Undergraduates

Companion to the paper by Eyal Weiss and Michael Margaliot
IEEE Transactions on Automatic Control, 2019

Abstract

This document explains every equation, definition, theorem, and algorithm in the paper "A Polynomial-Time Algorithm for Solving the Minimal Observability Problem in Conjunctive Boolean Networks" in detail, assuming only undergraduate-level knowledge of discrete mathematics and basic graph theory. No prior knowledge of Boolean networks, control theory, or observability is required. Each equation is accompanied by (1) a plain-English description of what it computes, (2) a symbol-by-symbol breakdown, (3) an intuitive explanation of why the computation makes sense, and (4) concrete numerical examples where helpful.

Contents

1 Background: Boolean Networks, State Variables, and Observability

1.1 What Is a State Variable?

Real-world systems—biological networks, social networks, power grids, epidemics—have many internal quantities that change over time. Each such quantity is called a **state variable** (SV).

For example, in a gene regulation network, each gene is either **ON** (expressed, value 1) or **OFF** (not expressed, value 0). The ON/OFF status of each gene is one state variable. If the network has 1000 genes, then the system has $n = 1000$ state variables, and at any moment the full state is a vector of 1000 zeros and ones.

Key Idea

A state variable is a single binary quantity (0 or 1) that describes one aspect of a system at a given moment in time. The full **state** of a system with n state variables is a vector of n binary values.

1.2 What Is a Boolean Network?

Let $S := \{0, 1\}$. A **Boolean network** (BN) with n state variables and m outputs is a discrete-time dynamical system. "Discrete-time" means the system updates at time steps $k = 0, 1, 2, 3, \ldots$ rather than continuously.

The dynamics are given by the paper's Equation (1):

$$\begin{aligned} X_i(k+1) &= f_i(X_1(k), \ldots, X_n(k)), \quad \forall i \in [1, n], \\ Y_j(k) &= h_j(X_1(k), \ldots, X_n(k)), \quad \forall j \in [1, m]. \end{aligned} \tag{1}$$

What it computes in plain English: At each time step k, every state variable X_i updates its value based on the current values of *all* state variables. Additionally, m **outputs** $Y_1, \ldots, Y_m$ are computed from the current state. The outputs are the quantities we can actually *measure* (observe with sensors).

Symbol-by-symbol breakdown:

- $X_i(k) \in \{0, 1\}$: the value of the i-th state variable at time step k.
- $k \in \{0, 1, 2, \ldots\}$: the discrete time index.
- $f_i : S^n \to S$: the **update function** (also called the Boolean function) for state variable i. It takes the current values of *all* n state variables and produces the new value of X_i.
- $[1, n] := \{1, 2, \ldots, n\}$: integer range notation.
- $Y_j(k) \in \{0, 1\}$: the j-th output at time k.
- $h_j : S^n \to S$: the output function for the j-th output.
- m: the number of outputs (sensors/measurements).

Why two lines? The first line describes how the system *evolves* (the dynamics). The second line describes what we can *see* (the measurements). The central challenge is that typically $m \ll n$: we observe far fewer quantities than the total number of state variables.

Numerical Example

Consider a tiny Boolean network with $n = 3$ state variables and $m = 1$ output:

$$\begin{aligned} X_1(k+1) &= X_2(k) \,\text{AND}\, X_3(k) \\ X_2(k+1) &= X_1(k) \\ X_3(k+1) &= X_2(k) \\ Y_1(k) &= X_1(k) \end{aligned}$$

Suppose the initial state is $X(0) = [1, 1, 0]'$. Then:

k	$X_1(k)$	$X_2(k)$	$X_3(k)$	$Y_1(k)$
0	1	1	0	1
1	$1 \,\text{AND}\, 0 = 0$	1	1	0
2	$1 \,\text{AND}\, 1 = 1$	0	1	1
3	$0 \,\text{AND}\, 1 = 0$	1	0	0

We can only observe the output sequence $\{Y_1(0), Y_1(1), Y_1(2), \ldots\} = \{1, 0, 1, 0, \ldots\}$. The question is: can we figure out the full initial state $X(0)$ from this output sequence?

1.3 What Is Observability?

Key Idea

A system is **observable** if, by watching only the outputs over some time window, you can always figure out what the initial state was. In other words, no two different initial states can produce the exact same output sequence.

Formally, denote the full state vector at time k by:

$$X(k) := \begin{bmatrix} X_1(k) & \cdots & X_n(k) \end{bmatrix}'$$

and the output vector by:

$$Y(k) := \begin{bmatrix} Y_1(k) & \cdots & Y_m(k) \end{bmatrix}'.$$

Observable on $[0, N]$: We say the BN is **observable on** $[0, N]$ if any two different initial conditions $X(0) \neq \tilde{X}(0)$ produce different output sequences $\{Y(0), \ldots, Y(N)\} \neq \{\tilde{Y}(0), \ldots, \tilde{Y}(N)\}$.

Observable: The BN is **observable** if it is observable on $[0, N]$ for some $N \geq 0$.

What does this mean practically? If the system is observable, then by collecting the output measurements $Y(0), Y(1), \ldots, Y(N)$, you can *uniquely reconstruct* the initial state $X(0)$. Once $X(0)$ is known, you can simulate the dynamics forward to determine $X(k)$ for *any* k. So observability means complete knowledge of the system's trajectory from limited measurements.

Indistinguishable states: If two different initial states $X(0)$ and $\tilde{X}(0)$ produce the *same* output sequence, they are called **indistinguishable**. The existence of indistinguishable states means the system is *not* observable: there is no way to tell them apart from outputs alone.

Directly Measurable vs. Observable

If there exists an output $Y_j(k) = X_i(k)$, then X_i is **directly measurable**: we read its value straight from the sensor at every time step. Being directly measurable is a very strong condition. Observability is weaker: a state variable does not need its own sensor;

its value can be inferred *indirectly* by watching how it affects other variables over time.

Numerical Example

Consider the 2-variable CBN from Example 1 in the paper:

$$X_1(k+1) = X_2(k),$$
$$X_2(k+1) = X_1(k) \cdot X_2(k).$$

Suppose we make X_1 directly measurable: $Y_1(k) = X_1(k)$.
Given $\{Y_1(0), Y_1(1)\}$, we know:

- $X_1(0) = Y_1(0)$ (directly measured at $k = 0$).
- $X_2(0) = X_1(1) = Y_1(1)$ (because the dynamics say $X_1(k+1) = X_2(k)$, so $X_1(1) = X_2(0)$).

So from just two output values, we recover the entire initial state. The system is observable on $[0, 1]$ with this single output.
Note that X_2 is never directly measured, but its initial value is determined *indirectly* through the dynamics.

1.4 What Is a Conjunctive Boolean Network (CBN)?

The paper focuses on a special class of Boolean networks where every update function uses only AND operations.

A BN is called a **conjunctive Boolean network** (CBN) if every update function includes AND operations only. The dynamics take the form of the paper's Equation (2):

$$X_i(k+1) = \prod_{j=1}^{n} (X_j(k))^{\epsilon_{ji}}, \quad \forall i \in [1, n], \tag{2}$$

where $\epsilon_{ji} \in \{0, 1\}$ for all i, j.

What it computes in plain English: The next value of X_i is the AND of a *subset* of the current state variables. The exponents ϵ_{ji} select which variables participate.

Symbol-by-symbol breakdown:

- $\prod$: product (multiplication). Since every $X_j(k)$ is either 0 or 1, the product of several such values equals 1 only when *all* of them are 1. This is exactly the AND operation.
- $(X_j(k))^{\epsilon_{ji}}$: if $\epsilon_{ji} = 1$, this contributes $X_j(k)$ to the product (variable X_j participates in the update of X_i). If $\epsilon_{ji} = 0$, this contributes $(X_j(k))^0 = 1$ (variable X_j does *not* affect X_i).
- $\epsilon_{ji} \in \{0, 1\}$: a binary indicator. It equals 1 iff X_j appears in the update function of X_i.

Why the product notation? Writing the AND function as a product is a compact way to handle a variable number of inputs. For example:

$$X_i(k+1) = X_2(k) \cdot X_5(k) \cdot X_7(k)$$

means X_i at the next time step equals 1 only if X_2, X_5, and X_7 are *all* 1 at the current time step. Otherwise, $X_i(k+1) = 0$.

Why AND is "Conservative"

In a CBN, each variable "wants" to be zero. It will only become (or stay) 1 if *all* its influencing variables are 1. A single zero among the influences forces the variable to zero. Think of it as a "conservative" or "pessimistic" update rule: every prerequisite must be met for the variable to be active.

Numerical Example

Consider a CBN with $n = 3$:

$$\begin{aligned} X_1(k+1) &= X_2(k) \cdot X_3(k), \\ X_2(k+1) &= X_1(k), \\ X_3(k+1) &= X_2(k). \end{aligned}$$

Here the exponents are:

ϵ_{ji}	$i=1$	$i=2$	$i=3$
$j=1$	0	1	0
$j=2$	1	0	1
$j=3$	1	0	0

So X_1's update depends on X_2 and X_3 (AND of both), X_2's update depends only on X_1 (identity), and X_3's update depends only on X_2.

Disjunctive Boolean Networks (DBNs)

A BN where every update function uses only OR is called a **disjunctive Boolean network** (DBN). By De Morgan's laws ($\overline{A \vee B} = \overline{A} \wedge \overline{B}$), every result about CBNs also applies to DBNs. So the paper's results cover both AND-only and OR-only networks.

1.5 The Dependency Graph

A key tool in the paper is representing a CBN as a **directed graph** (digraph).

Definition 1.1 (Dependency Graph)**.** Given a CBN of the form (2), the **dependency graph** $G = (V, E)$ is a directed graph where:

- $V = \{X_1, X_2, \ldots, X_n\}$ — one vertex per state variable.
- E contains a directed edge $e_{i \to j}$ (from X_i to X_j) if and only if $\epsilon_{ij} = 1$, i.e., X_i appears in the update function of X_j.

In plain English: Draw a node for each state variable. Draw an arrow from X_i to X_j whenever X_i directly influences the next value of X_j.

Important graph terminology:

- **In-neighbor** of X_j: a node X_i such that there is an edge $X_i \to X_j$. These are the variables that directly influence X_j's update. The set of in-neighbors is denoted $\mathcal{N}_{in}(X_j)$.
- **In-degree** of X_j: the number $|\mathcal{N}_{in}(X_j)|$ of edges pointing *into* X_j.
- **Out-neighbor** of X_i: a node X_j such that there is an edge $X_i \to X_j$. These are the variables that X_i directly influences. Denoted $\mathcal{N}_{out}(X_i)$.
- **Source**: a node with in-degree zero (nothing feeds into it).

- **Sink**: a node with out-degree zero (it does not influence anything else).
- **Walk**: a sequence of nodes connected by edges.
- **Path**: a walk with no repeated vertices.
- **Cycle**: a closed walk where all vertices are distinct except the start and end vertex.
- **Reachable**: X_i is reachable from X_j if there is a directed path from X_j to X_i.

Key Idea

There is a one-to-one correspondence between a CBN and its dependency graph. Knowing the dependency graph is equivalent to knowing all the update functions. This allows us to analyze observability as a *graph problem*.

Numerical Example

For the CBN:

$$\begin{aligned} X_1(k+1) &= X_2(k) \cdot X_3(k), \\ X_2(k+1) &= X_1(k), \\ X_3(k+1) &= X_2(k), \end{aligned}$$

the dependency graph has:

- Edge $X_2 \to X_1$ (because X_2 appears in X_1's update).
- Edge $X_3 \to X_1$ (because X_3 appears in X_1's update).
- Edge $X_1 \to X_2$ (because X_1 appears in X_2's update).
- Edge $X_2 \to X_3$ (because X_2 appears in X_3's update).

Notice: X_1 has in-degree 2 (two arrows point into it), X_2 has in-degree 1, and X_3 has in-degree 1. The in-neighbor sets are: $\mathcal{N}_{in}(X_1) = \{X_2, X_3\}$, $\mathcal{N}_{in}(X_2) = \{X_1\}$, $\mathcal{N}_{in}(X_3) = \{X_2\}$.

1.6 The Minimal Observability Problem

When a system is not observable, we can try to *add sensors* (outputs) to make it observable. Each additional output directly measures one state variable. The **minimal observability problem** asks: what is the fewest number of sensors we need to add?

Definition 1.2 (Problem 1 from the paper)**.** Given a CBN with n SVs, determine a minimal set of indices $\mathcal{I} \subseteq [1, n]$, such that making each $X_i(k)$, $i \in \mathcal{I}$, directly measurable yields an observable CBN.

In plain English: Find the smallest set of state variables that need their own sensors so that the entire system becomes observable.

Why does this matter?

1. **Practical:** Sensors are expensive (in cost, in invasiveness for biological systems, etc.). We want to place as few as possible.
2. **Theoretical:** The solution identifies the "most informative" nodes—the ones whose measurement gives the most information about the entire system.

Agent Opinion Network

Consider a social network of agents, each holding an opinion (0 or 1) at time k. Each agent is "conservative": they adopt opinion 1 only if *all* their neighbors hold opinion 1. This is modeled as a CBN.
We want to recruit the fewest possible agents as informants (adding sensors on their opinions) so that we can reconstruct everyone's opinion trajectory. The minimal observability problem tells us exactly which agents to recruit.

Epidemic Monitoring

Consider agents that can be susceptible (0) or infected (1). If a single neighbor of an agent is infected, the agent becomes infected. This is a disjunctive BN (OR dynamics), which by De Morgan's laws can be reduced to a CBN problem.
The minimal observability problem tells us the fewest people to test so that we can reconstruct the entire infection history.

2 Necessary Conditions for Observability

Before solving the minimal observability problem, the paper first answers: *what structural properties must a CBN's dependency graph have for the CBN to be observable?*

Throughout this section, we consider CBNs with n SVs and $m \geq 0$ outputs in the simplified form from the paper's Equation (3):

$$\begin{aligned} X_i(k+1) &= f_i(X_1(k), \ldots, X_n(k)), \qquad \forall i \in [1, n], \\ Y_j(k) &= X_j(k), \qquad \forall j \in [1, m], \end{aligned} \tag{3}$$

where the f_is are AND operators, and every output Y_j is the value of the j-th SV. (The first m nodes are directly observable; nodes $X_{m+1}, \ldots, X_n$ are not.)

2.1 Property O_1: Every Non-Observable Node Must Be Uniquely Responsible for Some Update

Definition 2.1 (Property O_1)**.** A CBN has **Property** O_1 if for every non-directly observable node X_i there exists some other node X_j such that $\mathcal{N}_{in}(X_j) = \{X_i\}$.

In plain English: Every "hidden" (non-observed) node must be the *sole influencer* of at least one other node. That is, there must exist some node X_j whose update function depends on X_i and *only* on X_i: $X_j(k+1) = X_i(k)$.

Why is this necessary? If a hidden node X_i is *not* the sole influencer of any other node, then we can never isolate its value. In a CBN, setting one variable to zero "kills" all AND functions it participates in. So if X_i always appears in AND combinations with other variables, its contribution is masked whenever any of those other variables is zero.

Fact 2.1. *If a CBN is observable then it has Property* O_1.

Proof idea: The proof considers two cases where Property O_1 fails:

1. **Case 1 — X_i is a sink:** If X_i has no outgoing edges and is not directly observed, there is simply no way to determine $X_i(0)$ from the outputs. It never influences anything we can see.

2. **Case 2 — X_i always shares influence:** If every node X_j that X_i feeds into also receives input from other nodes, then consider two initial conditions:

   - All zeros: $X(0) = [0, 0, \ldots, 0]'$.
   - All zeros except $X_i(0) = 1$: $\tilde{X}(0) = [0, \ldots, 0, 1, 0, \ldots, 0]'$.

   In both cases, every AND function that involves X_i *also* involves at least one other variable that is zero, so the output is zero regardless. The two states produce identical output sequences—they are indistinguishable.

Numerical Example

Consider the CBN from Example 2 in the paper:

$$\begin{aligned} X_1(k+1) &= X_2(k) \cdot X_3(k), \\ X_2(k+1) &= X_1(k), \\ X_3(k+1) &= X_2(k), \\ Y_1(k) &= X_1(k). \end{aligned}$$

Node X_1 is directly observable. Let us check Property O_1 for the hidden nodes:

- X_2 is hidden. Is there a node whose *only* in-neighbor is X_2? Yes: $\mathcal{N}_{in}(X_3) = \{X_2\}$. So X_2 satisfies the condition.
- X_3 is hidden. Is there a node whose *only* in-neighbor is X_3? We need $\mathcal{N}_{in}(X_j) = \{X_3\}$ for some j. Check all nodes: $\mathcal{N}_{in}(X_1) = \{X_2, X_3\}$ (not just X_3), $\mathcal{N}_{in}(X_2) = \{X_1\}$ (not X_3), $\mathcal{N}_{in}(X_3) = \{X_2\}$ (not X_3). No such node exists!

X_3 violates Property O_1. Verify: the initial conditions $X(0) = [0, 0, 0]'$ and $\tilde{X}(0) = [0, 0, 1]'$ both produce $Y_1(k) = 0$ for all $k \geq 0$. These states are indistinguishable, so the CBN is **not** observable.

2.2 Property O_2: Cycles of Hidden Nodes Must Have an "Escape Route"

Definition 2.2 (Property O_2)**.** A CBN has **Property** O_2 if for every cycle C in its dependency graph that is composed solely of non-directly observable nodes, the following holds: C includes a node X_i which is the only element in the in-neighbors' set of some other node X_j, i.e., $\mathcal{N}_{in}(X_j) = \{X_i\}$, and X_j is *not* part of the cycle C.

In plain English: If you have a cycle of hidden nodes (they keep feeding into each other in a loop), at least one node in that cycle must "point to" a node *outside* the cycle as that outside node's sole influencer. This gives the cycle an "escape route" through which information can leak out to eventually reach an output.

Why is this necessary? Consider a cycle of hidden nodes $X_{k_1} \to X_{k_2} \to \cdots \to X_{k_\ell} \to X_{k_1}$ where none of them has a unique outgoing "pointer" to a node outside the cycle. Then the values on this cycle circulate among themselves without ever uniquely influencing anything observable. Setting all variables to zero, versus setting one cycle variable to one (and all others to zero), produces the same output: zero everywhere.

Fact 2.2. *If a CBN is observable then it satisfies Property O_2.*

Proof idea: If Property O_2 fails, the dependency graph has a cycle C of hidden nodes where none of the cycle nodes is the sole in-neighbor of any node outside C. Compare the all-zeros initial condition with the initial condition that has one cycle node set to 1. In the all-zeros case, every AND function involving a cycle node outputs zero (the other inputs are zero). In the one-on-cycle case, the value 1 circulates within the cycle. But because no cycle node is the

sole influencer of any outside node, the 1 never "escapes" to affect any directly observable node uniquely—the AND with other zero inputs kills it. The output sequences are identical, so the states are indistinguishable.

Numerical Example

Consider the CBN from Example 3 in the paper (6 nodes, 1 output):

$$\begin{aligned} X_1(k+1) &= X_2(k) \cdot X_4(k), \\ X_2(k+1) &= X_3(k), \\ X_3(k+1) &= X_2(k), \\ X_4(k+1) &= X_6(k), \\ X_5(k+1) &= X_4(k), \\ X_6(k+1) &= X_5(k), \\ Y_1(k) &= X_1(k). \end{aligned}$$

Only X_1 is directly observable. Check Property O_1:

- X_2: $\mathcal{N}_{in}(X_3) = \{X_2\}$. OK.
- X_3: $\mathcal{N}_{in}(X_2) = \{X_3\}$. OK.
- X_4: $\mathcal{N}_{in}(X_5) = \{X_4\}$. OK.
- X_5: $\mathcal{N}_{in}(X_6) = \{X_5\}$. OK.
- X_6: $\mathcal{N}_{in}(X_4) = \{X_6\}$. OK.

Property O_1 holds! But now check Property O_2. The cycle $X_4 \to X_5 \to X_6 \to X_4$ consists entirely of hidden nodes. Does any node in this cycle point uniquely to a node *outside* the cycle?

- X_4: $\mathcal{N}_{in}(X_5) = \{X_4\}$, but X_5 is *inside* the cycle. Does not count. Also X_4 feeds into X_1, but $\mathcal{N}_{in}(X_1) = \{X_2, X_4\}$ (not sole influencer).
- X_5: $\mathcal{N}_{in}(X_6) = \{X_5\}$, but X_6 is inside the cycle. Does not count.
- X_6: $\mathcal{N}_{in}(X_4) = \{X_6\}$, but X_4 is inside the cycle. Does not count.

Property O_2 fails! Indeed, the initial conditions $X(0) = [0,0,0,0,0,0]'$ and $\tilde{X}(0) = [0,0,0,1,1,1]'$ both produce $Y_1(k) = 0$ for all $k \geq 0$. (In the second case, the 1s circulate within the $\{X_4, X_5, X_6\}$ cycle, but the AND at X_1 also requires $X_2 = 1$, which stays zero.) The CBN is **not** observable.

3 The Main Theorem: Necessary and Sufficient Conditions

3.1 Theorem 1: The Complete Characterization

The paper's central theoretical result combines Properties O_1 and O_2 into a complete characterization:

Theorem 3.1 (Theorem 1 from the paper)**.** *A CBN is observable if and only if it satisfies Properties O_1 and O_2.*

In plain English: Properties O_1 and O_2 are not only *necessary* (as shown above) but also *sufficient*. If both properties hold, the CBN is guaranteed to be observable. No other conditions are needed.

Why this is surprising: For *general* Boolean networks, testing observability is NP-hard (computationally intractable for large systems). But for CBNs, observability reduces to two simple graph-theoretic properties that can be checked efficiently. This is a dramatic simplification.

3.2 Observed Paths: The Key Concept for the Proof

To prove Theorem 1, the paper introduces the concept of an **observed path**.

Definition 3.1 (Observed Path)**.** An **observed path** in the dependency graph is a non-empty ordered set of nodes such that:

1. The **last** element is a directly observable node.
2. If the set contains $p > 1$ elements, then for any $i < p$, the i-th element is a non-directly observable node and is the *only element* in the in-neighbors' set of the $(i+1)$-th element.

In plain English: An observed path is a chain of nodes

$$X_{a_1} \to X_{a_2} \to \cdots \to X_{a_{p-1}} \to X_{a_p}$$

where:

- X_{a_p} (the end) is directly observable.
- Each intermediate node X_{a_i} (for $i < p$) is hidden and is the *sole* input to $X_{a_{i+1}}$: $\mathcal{N}_{in}(X_{a_{i+1}}) = \{X_{a_i}\}$.

Observed Paths as Shift Registers

An observed path works like a **shift register**: each node copies its value to the next node at each time step (since $\mathcal{N}_{in}(X_{a_{i+1}}) = \{X_{a_i}\}$ means $X_{a_{i+1}}(k+1) = X_{a_i}(k)$).
If the path has length p, then the value of the *first* node at time 0 will appear at the *last* (observable) node at time $p-1$. By watching the output over $p-1$ time steps, we can read off the initial values of every node in the path, in reverse order.

Numerical Example

Consider an observed path $X_4 \to X_3 \to X_1$ where X_1 is directly observable and $\mathcal{N}_{in}(X_3) = \{X_4\}$, $\mathcal{N}_{in}(X_1) = \{X_3\}$.
The dynamics along this path are:

$$X_3(k+1) = X_4(k),$$
$$X_1(k+1) = X_3(k).$$

Reading the output $Y_1(k) = X_1(k)$:

- $Y_1(0) = X_1(0)$ gives us $X_1(0)$ directly.
- $Y_1(1) = X_1(1) = X_3(0)$ gives us $X_3(0)$.
- $Y_1(2) = X_1(2) = X_3(1) = X_4(0)$ gives us $X_4(0)$.

From three output values, we recover the initial values of all three nodes in the path. The path has length 3, and we need $3 - 1 = 2$ time steps.

3.3 Decomposition into Disjoint Observed Paths

Proposition 3.2 (Proposition 1 from the paper)**.** *Consider a CBN with a dependency graph G that satisfies Properties O_1 and O_2. Then G can be decomposed into **disjoint observed paths** such that every vertex in the graph belongs to a single observed path.*

In plain English: If both properties hold, we can partition all nodes into non-overlapping chains, each ending at an observable node. Every node belongs to exactly one chain.

Corollary 3.3 (Corollary 1 from the paper)**.** *A CBN is observable if and only if its dependency graph can be decomposed into a set of disjoint observed paths.*

Why this gives observability: If every node belongs to an observed path, then every node's initial value can be read from some output after a certain number of time steps. The required observation window is $[0, \max_i\{N_i\} - 1]$, where N_i is the length of the i-th observed path.

3.4 Algorithm 1: Decomposing into Observed Paths

The paper provides an explicit algorithm (Algorithm 1) for decomposing a dependency graph into disjoint observed paths. Here is the step-by-step logic:

1. **For each directly observable node** X_i $(i = 1, \ldots, m)$: start a new path ending with X_i.
2. **Trace backwards:** Look at X_i's in-neighbor. If X_i has exactly one in-neighbor v ($|\mathcal{N}_{in}(X_i)| = 1$), and v is hidden and not yet assigned to a path, then prepend v to the path.
3. **Repeat:** Now treat v as the current node and repeat step 2: check if v has exactly one in-neighbor that is hidden and unassigned. If so, prepend it. Continue until no more nodes can be added.
4. **Output the path:** Print the completed observed path and move to the next directly observable node.

Numerical Example

Consider the CBN from Example 4 in the paper (5 nodes, 2 outputs):

$$\begin{aligned} X_1(k+1) &= X_3(k), \\ X_2(k+1) &= X_5(k), \\ X_3(k+1) &= X_4(k), \\ X_4(k+1) &= X_2(k) \cdot X_3(k), \\ X_5(k+1) &= X_1(k) \cdot X_5(k), \\ Y_1(k) &= X_1(k), \\ Y_2(k) &= X_2(k). \end{aligned}$$

Directly observable: X_1, X_2.

Path starting from X_1:

- Start: path $= \{X_1\}$.
- $\mathcal{N}_{in}(X_1) = \{X_3\}$ (size 1). X_3 is hidden and unassigned. Prepend: path $= \{X_3, X_1\}$.
- $\mathcal{N}_{in}(X_3) = \{X_4\}$ (size 1). X_4 is hidden and unassigned. Prepend: path $= \{X_4, X_3, X_1\}$.
- $\mathcal{N}_{in}(X_4) = \{X_2, X_3\}$ (size 2). Stop.
- Output path $O^1 = (X_4, X_3, X_1)$.

Here $X_4 \mapsto X_3 \mapsto X_1$, meaning X_3's sole input is X_4, and X_1's sole input is X_3.

Path starting from X_2:

- Start: path $= \{X_2\}$.
- $\mathcal{N}_{in}(X_2) = \{X_5\}$ (size 1). X_5 is hidden and unassigned. Prepend: path $= \{X_5, X_2\}$.
- $\mathcal{N}_{in}(X_5) = \{X_1, X_5\}$ (size > 1, counting the self-loop). Stop.
- Output path $O^2 = (X_5, X_2)$.

Every node is in exactly one path. To reconstruct all initial values:

- From $O^1 = (X_4, X_3, X_1)$: $Y_1(0) = X_1(0)$, $Y_1(1) = X_3(0)$, $Y_1(2) = X_4(0)$.
- From $O^2 = (X_5, X_2)$: $Y_2(0) = X_2(0)$, $Y_2(1) = X_5(0)$.

The observation window is $[0, 2]$ (length of the longest path minus 1).

3.5 Observer Design

The proof of Theorem 1 also gives an explicit procedure for designing an **observer**—a device that reconstructs the initial state from the output measurements.

Observer design procedure:

1. Construct the dependency graph G.
2. Apply Algorithm 1 to decompose G into disjoint observed paths $O^1, \ldots, O^m$.
3. Observe the output sequence for $N = \max_i |O^i| - 1$ time steps.
4. For each observed path $O^i = (X_{a_1}, \ldots, X_{a_{N_i}})$ with $X_{a_{N_i}}$ directly observable: read the initial values as $X_{a_j}(0) = Y_i(N_i - j)$ for $j = 1, \ldots, N_i$.

In plain English: The output of each sensor, read backwards in time, directly gives you the initial values of the nodes in its observed path, from the end of the path to the beginning.

4 The Algorithm for Minimal Observability (Algorithm 2)

This section describes the paper's main algorithmic contribution: an efficient algorithm that solves Problem 1 (the minimal observability problem) in $O(n^2)$ time.

4.1 High-Level Description

Algorithm 2 determines the minimal set of SVs that must be made directly observable so that the resulting CBN is observable. The idea is to find exactly which nodes violate Properties O_1 and O_2, and add the minimum number of sensors to fix these violations.

The algorithm proceeds in 7 steps:

1. **Generate the dependency graph** $G = (V, E)$.
2. **Build list** L_1**:** All SVs that are *not* directly observable and are *not* the only element in the in-neighbors' set of another node. These are the nodes that violate Property O_1—they need to become directly observable.
3. **Build list** L_2**:** All SVs that are *not* directly observable and *are* the only element in the in-neighbors' set of another node. These nodes satisfy Property O_1 individually, but they might be part of problematic cycles.
4. **Find problematic cycles (**L_C**):** Create a list of all cycles composed solely of nodes in L_2.
5. **Check Property** O_2 **for each cycle:** For each cycle $C \in L_C$, check if one of its elements appears as the sole in-neighbor of a node *not* in C. If so, Property O_2 is satisfied for this cycle, and it is removed from L_C.
6. **Build the output set** $\mathcal{I}$**:** Start with $\mathcal{I} = L_1$ (all nodes violating O_1). Then, for each remaining cycle in L_C (those violating O_2), pick *one* representative node from the cycle and add it to $\mathcal{I}$.
7. **Return** $\mathcal{I}$**:** This is the minimal set of SVs to add as outputs.

Why One Node per Cycle Suffices

When a cycle of hidden nodes violates Property O_2, making *one* node in the cycle directly observable "breaks" the cycle: the cycle no longer consists entirely of hidden nodes. The

remaining nodes in the cycle can form an observed path ending at the newly observed node. So one sensor per problematic cycle is both necessary and sufficient.

4.2 Detailed Walkthrough of Algorithm 2

The detailed algorithm (given in the paper's Appendix) uses the following data structures:

- L_1: a bit-list of length n. $L_1(i) = 1$ means node X_i violates Property O_1 and must become directly observable.
- L_2: a bit-list of length n. $L_2(i) = 1$ means node X_i is the sole in-neighbor of some other node (a candidate for observed-path building).
- L_{pairs}: an $n \times 2$ matrix. If $\mathcal{N}_{in}(X_i) = \{X_j\}$ (node X_i has in-degree one with sole in-neighbor X_j), then $L_{pairs}(i, 1) = 1$ and $L_{pairs}(i, 2) = j$. This records that X_i has exactly one incoming edge, coming from X_j.
- L_3: a bit-list derived from L_2 that, after a reduction process, contains exactly the nodes forming problematic cycles.

Steps 3–5 (in the Appendix's numbering) build L_2 and L_{pairs}: for each node X_i with $|\mathcal{N}_{in}(X_i)| = 1$, if the sole in-neighbor X_j is hidden ($j \neq i$), mark $L_2(j) = 1$ and record $L_{pairs}(i, 1) = 1$, $L_{pairs}(i, 2) = j$.

Steps 6–7 build L_1 from the nodes not in L_2 and not directly observable.

Steps 9–15 reduce L_2 to L_3 by a depth-first-search-like tracing. Starting from nodes *not* in L_3 (those that are directly observable or in L_1), trace through the "pointing" pairs to remove nodes from L_3 that can be reached from a non-cycle starting point. After this reduction, L_3 contains exactly the nodes that are part of problematic cycles.

Steps 16–18: Build a subgraph $\tilde{G}$ from the remaining nodes in L_3. Every connected component of $\tilde{G}$ is a cycle (since every node in L_3 has exactly in-degree one and out-degree one within $\tilde{G}$). Find the cycles using a strongly connected components algorithm, pick one representative from each, and add them to $\mathcal{I}$.

4.3 Worked Example: Algorithm 2 on Example 2's CBN

Algorithm 2 on Example 2

Recall the CBN from Example 2:

$$\begin{aligned} X_1(k+1) &= X_2(k) \cdot X_3(k), \\ X_2(k+1) &= X_1(k), \\ X_3(k+1) &= X_2(k), \\ Y_1(k) &= X_1(k). \end{aligned}$$

$m = 1$, so X_1 is directly observable. Hidden nodes: X_2, X_3.
Step 1: Build dependency graph. Edges: $X_2 \to X_1$, $X_3 \to X_1$, $X_1 \to X_2$, $X_2 \to X_3$.
Step 2 (Build L_1): Check which hidden nodes are NOT the sole in-neighbor of any node.

- X_2: Is X_2 the sole in-neighbor of any node? $\mathcal{N}_{in}(X_3) = \{X_2\}$. Yes! So $X_2 \notin L_1$.
- X_3: Is X_3 the sole in-neighbor of any node? $\mathcal{N}_{in}(X_1) = \{X_2, X_3\}$. No (not sole). No other node has X_3 as sole in-neighbor. So $X_3 \in L_1$.

$L_1 = \{X_3\}$.
Step 3 (Build L_2): Hidden nodes that ARE the sole in-neighbor of some node: $L_2 = \{X_2\}$.
Step 4 (Find cycles in L_2): $L_2 = \{X_2\}$. A single node cannot form a cycle (no self-loop on X_2). $L_C = \emptyset$.
Step 6 (Build $\mathcal{I}$): $\mathcal{I} = L_1 = \{X_3\}$. No cycles to handle.
Result: Make X_3 directly observable. The modified CBN has outputs $Y_1(k) = X_1(k)$ and $Y_2(k) = X_3(k)$, and it is observable.
Verification: After adding $Y_2(k) = X_3(k)$, we have two outputs. The observed paths are:

- O^1: Starting from X_1, trace back. $\mathcal{N}_{in}(X_1) = \{X_2, X_3\}$ (size 2). Stop. Path: (X_1).
- O^2: Starting from X_3, trace back. $\mathcal{N}_{in}(X_3) = \{X_2\}$ (size 1, hidden). Prepend. Path: (X_2, X_3).

All three nodes covered. Observable on $[0, 1]$: $Y_1(0) = X_1(0)$, $Y_2(0) = X_3(0)$, $Y_2(1) = X_2(0)$. Since 1 output was already given and Algorithm 2 added only 1 more, and observability requires at least 1 output, this is minimal.

Algorithm 2 on Example 3

Recall the CBN from Example 3 (6 nodes, 1 output):

$$\begin{aligned} &X_1(k+1) = X_2(k) \cdot X_4(k), \quad X_2(k+1) = X_3(k), \quad X_3(k+1) = X_2(k), \\ &X_4(k+1) = X_6(k), \quad X_5(k+1) = X_4(k), \quad X_6(k+1) = X_5(k), \\ &\qquad Y_1(k) = X_1(k). \end{aligned}$$

Only X_1 is directly observable. Hidden: X_2, X_3, X_4, X_5, X_6.
Step 2 (Build L_1): Check each hidden node—is it the sole in-neighbor of any node?

- X_2: $\mathcal{N}_{in}(X_3) = \{X_2\}$. Yes. $X_2 \notin L_1$.
- X_3: $\mathcal{N}_{in}(X_2) = \{X_3\}$. Yes. $X_3 \notin L_1$.
- X_4: $\mathcal{N}_{in}(X_5) = \{X_4\}$. Yes. $X_4 \notin L_1$.
- X_5: $\mathcal{N}_{in}(X_6) = \{X_5\}$. Yes. $X_5 \notin L_1$.
- X_6: $\mathcal{N}_{in}(X_4) = \{X_6\}$. Yes. $X_6 \notin L_1$.

$L_1 = \emptyset$ (all hidden nodes satisfy O_1 individually).
Step 3 (Build L_2): $L_2 = \{X_2, X_3, X_4, X_5, X_6\}$.
Step 4 (Find cycles in L_2):

- $X_2 \to X_3 \to X_2$: a cycle of length 2 in L_2. Call it C_1.
- $X_4 \to X_5 \to X_6 \to X_4$: a cycle of length 3 in L_2. Call it C_2.

$L_C = \{C_1, C_2\}$.
Step 5 (Check Property O_2):

- $C_1 = \{X_2, X_3\}$: Does any node in C_1 uniquely point to a node outside C_1? X_2 points uniquely to X_3 (inside C_1). Also, X_2 feeds into X_1 but $\mathcal{N}_{in}(X_1) = \{X_2, X_4\}$ (not sole). X_3 points uniquely to X_2 (inside C_1). No escape route found. C_1 stays in L_C.
- $C_2 = \{X_4, X_5, X_6\}$: Similarly, all "sole pointing" stays within the cycle. C_2 stays in L_C.

Step 6 (Build $\mathcal{I}$): $\mathcal{I} = L_1 \cup \{\text{one from each cycle}\} = \emptyset \cup \{X_2, X_4\} = \{X_2, X_4\}$ (choosing one representative per cycle).
Result: Add outputs $Y_2(k) = X_2(k)$ and $Y_3(k) = X_4(k)$. Total outputs: $Y_1 = X_1$, $Y_2 = X_2$, $Y_3 = X_4$. Two additional sensors are needed. This is minimal because there are two independent problematic cycles, each requiring at least one sensor.

4.4 Correctness of Algorithm 2 (Theorem 2)

Theorem 4.1 (Theorem 2 from the paper)**.** *Algorithm 2 provides a solution to Problem 1 (the minimal observability problem).*

Why the output is correct (observability):

- Step 2 catches all nodes violating Property O_1 and makes them directly observable, restoring O_1.
- Steps 4–5 identify all cycles violating Property O_2.
- Step 6 adds one sensor per violating cycle, breaking each cycle and restoring O_2.

After adding the sensors in $\mathcal{I}$, both O_1 and O_2 hold, so by Theorem 1 the modified CBN is observable.

Why the output is minimal:

- Every node in L_1 *must* become observable: no other modification can fix its O_1 violation (making other nodes observable does not change the dependency graph structure).
- Every cycle in L_C requires at least one of its nodes to become observable (otherwise the cycle of hidden nodes remains, violating O_2).
- Different cycles in L_C are disjoint (they share no nodes, since they are composed of L_2 nodes which form distinct cycles), so observing one cycle's node does not help another cycle.

Therefore $|L_1|+|L_C|$ is a lower bound on the number of needed sensors, and Algorithm 2 achieves exactly this.

4.5 Complexity Analysis

Running Time: $O(n^2)$

The algorithm runs in polynomial time, specifically $O(n^2)$ where n is the number of state variables.

Step-by-step complexity:

1. **Generating the dependency graph:** Requires examining each of the n update functions, each with at most n arguments. Total: $O(n^2)$. The resulting graph has $|V| = n$ and $|E| \leq n^2$.
2. **Building L_1, L_2, L_{pairs}:** For each node, check its in-degree. This is $O(|V|+|E|) = O(n^2)$.
3. **Building L_3 (the reduction):** The depth-first tracing visits each node at most once. Total: $O(n)$.
4. **Finding cycles in $\tilde{G}$:** Using a strongly connected components algorithm (e.g., Tarjan's), which runs in $O(|V| + |E|)$. Since $\tilde{G}$ is a subgraph of G: $O(n^2)$ in the worst case, but typically much smaller.
5. **Overall:** $O(n^2)$.

Practical running times (from the paper's experiments):

n (number of SVs)	Running time
1,000	≈ 0.03 seconds
10,000	≈ 2.8 seconds

These times were measured using MATLAB on a standard PC (Intel Core i5, 4 GB RAM).

Contrast with General Boolean Networks

For *general* BNs (not restricted to conjunctive functions), testing observability is **NP-hard**. This means there is no known polynomial-time algorithm, and finding one would prove P = NP. The restriction to AND-only (or OR-only) update functions is what makes the problem tractable.

5 Minimal Observability in Random CBNs

The paper applies the algorithm to random CBNs, where the dependency graph is a **directed Erdős–Rényi random graph**.

5.1 The Random Graph Model

Fix n (number of vertices) and a probability $p \in [0, 1]$. Each possible directed edge is included independently with probability p. This gives a random dependency graph, and hence a random CBN.

What does p control?

- Small p ($p \to 0$): very sparse graph (few edges). Most nodes have small in-degree, so many nodes have in-degree exactly 1.
- Large p ($p \to 1$): very dense graph (many edges). Most nodes have large in-degree, so few nodes have in-degree exactly 1.
- The interesting regime turns out to be $p \approx 1/n$.

5.2 The Key Quantity: Nodes with In-Degree One

The paper identifies the set W of vertices with in-degree exactly one and no self-loops as the key quantity for the analysis.

Why does W matter? A node X_i with in-degree one means $|\mathcal{N}_{in}(X_i)| = 1$—there is exactly one node pointing into X_i. This sole in-neighbor can potentially be part of an observed path. In the observed path decomposition, only nodes in W can be "interior" nodes of a path (non-first nodes), because being in a path requires being the sole in-neighbor of the next node. Nodes *not* in W can only be the *first* node of an observed path and thus require their own output.

Probability that a node belongs to W:

$$q(p) := (n-1)p(1-p)^{n-1}. \tag{$q(p)$}$$

Symbol-by-symbol breakdown:

- p: the probability that any given edge exists.
- $(1-p)^{n-1}$: the probability that $n-1$ specific potential edges are *all absent*. Specifically, the self-loop must be absent (one factor of $(1-p)$) and the $n-2$ other non-chosen incoming edges must also be absent (contributing $(1-p)^{n-2}$), giving $(1-p)^1 \cdot (1-p)^{n-2} = (1-p)^{n-1}$. This is exact, not an approximation.
- $(n-1)p$: choosing which one of the $n-1$ possible in-edges exists, times its probability. More precisely, there are $\binom{n-1}{1} = n-1$ ways to choose exactly one edge from $n-1$ possible edges.

- $q(p)$: the probability that a given node has in-degree exactly one and no self-loop.

Expected size of W: Since each of the n nodes independently belongs to W with probability $q(p)$:

$$\mathbb{E}[|W|] = q(p) \cdot n = (n-1)p(1-p)^{n-1} \cdot n.$$

Numerical Example

With $n = 1000$ and $p = 1/1000 = 0.001$:

$$\begin{aligned} q(0.001) &= 999 \cdot 0.001 \cdot (1 - 0.001)^{999} \\ &= 0.999 \cdot (0.999)^{999} \\ &\approx 0.999 \cdot e^{-0.999} \\ &\approx 0.999 \cdot 0.368 \\ &\approx 0.368. \end{aligned}$$

So about 36.8% of nodes have in-degree exactly one—these can potentially be interior nodes of observed paths. The remaining $\approx$ 63.2% need their own output sensors (at minimum).

5.3 Lower and Upper Bounds on the Number of Outputs

Let k denote the minimal number of outputs needed to make the random CBN observable.

Lower bound (Equation 7): Every node *not* in W must be the first node of an observed path, requiring its own output. Therefore $k \geq n - |W|$, giving:

$$\frac{100}{n}\mathbb{E}[k] \geq 100\frac{n - \mathbb{E}[|W|]}{n} = \underline{s}, \tag{7}$$

where $\underline{s}(p) := 100(1-q(p))$ is the lower bound on the percentage of nodes that must be observed.

Upper bound (Equation 8): Nodes in W can form cycles of W-nodes, and each such cycle requires one additional output. Since the shortest possible cycle has length 2, the maximum number of cycles among $|W|$ nodes is $|W|/2$. Therefore:

$$\frac{100}{n}\mathbb{E}[k] \leq 100\frac{n - \mathbb{E}[|W|]/2}{n} = \bar{s}, \tag{8}$$

where $\bar{s}(p) := 100(1 - q(p)/2)$ is the upper bound.

In plain English:

- **Best case** (lower bound): all nodes in W line up into long chains without forming cycles. Then $k = n - |W|$.
- **Worst case** (upper bound): all nodes in W pair up into 2-cycles. Each 2-cycle requires one sensor, so $k = n - |W| + |W|/2 = n - |W|/2$.

Numerical Example

With $n = 1000$ and $p = 0.001$ (so $q \approx 0.368$):

- Lower bound: $\underline{s} = 100(1 - 0.368) = 63.2\%$. At least 632 of 1000 nodes must be observed.
- Upper bound: $\bar{s} = 100(1 - 0.368/2) = 100(1 - 0.184) = 81.6\%$. At most 816 nodes need to be observed.

The simulation result of $s^* \approx 69.3\%$ falls between these bounds.

5.4 The Optimal Edge Probability

The probability $q(p)$ is maximized when the number of in-degree-one nodes is largest, which minimizes the number of needed outputs.

Taking the derivative and setting it to zero:

$$\frac{dq}{dp} = (n-1)(1-p)^{n-2}(1-np) = 0$$

gives the unique maximum at:

$$p^* := n^{-1} = \frac{1}{n}.$$

In plain English: The optimal edge probability is $1/n$. At this density, the graph has just enough edges for maximum information flow without overwhelming nodes with too many inputs (which would prevent them from having in-degree one).

At $p^* = 1/n$:

$$q(p^*) = (n-1) \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \approx 1 \cdot e^{-1} \approx 0.368.$$

And the lower bound on the percentage of needed outputs is (Equation 9):

$$\underline{s}(p^*) = 100(1 - e^{-1}) \approx 63.2\%. \tag{9}$$

The 70% Rule

Even in the best case (optimal graph topology), at least about 63% of nodes must be observed, and in practice (from simulations) about 70% must be observed. This means that for random CBNs, observability is inherently "expensive"—you need sensors on the majority of nodes.

Numerical Example

Why $p = 1/n$ is a sweet spot:
Consider $n = 100$.

- $p = 1/10000$: Very sparse. Most nodes have in-degree 0 (sources). Sources cannot be part of any observed path as interior nodes. Almost all nodes need their own sensor: $\underline{s} \approx 99\%$.
- $p = 1/100 = 1/n$: Sweet spot. $q \approx 0.37$, so $\underline{s} \approx 63\%$.
- $p = 1/10$: Dense. Most nodes have in-degree $\approx 10 \gg 1$. Very few nodes have in-degree exactly 1. Almost all nodes need their own sensor: $\underline{s} \approx 99.97\%$.

At $p = 1/n$, the expected in-degree of each node is $(n-1)/n \approx 1$, which maximizes the chance of getting in-degree exactly 1.

5.5 The Sharp Transition

The paper computes the second derivative of $q(p)$ at $p = p^*$:

$$\left|\frac{d^2q}{dp^2}\right|_{p=p^*} = (n-1)^2(1-n^{-1})^{n-3} \approx \frac{n^3}{n-1}e^{-1}.$$

This is $\Theta(n^2)$ for large n, which means the curve $q(p)$ has extremely high curvature at the maximum.

In plain English: The function $q(p)$ (and hence the bounds $\underline{s}$ and $\bar{s}$) changes very rapidly near $p = 1/n$. Even a small deviation from the optimal probability dramatically increases the number of needed sensors. This is a **phase-transition-like behavior**: the graph is finely tuned at $p = 1/n$, and performance degrades sharply on either side.

This phenomenon is reminiscent of the phase transition in *undirected* Erdős–Rényi graphs, where the size of the largest connected component undergoes a sharp transition at $p = 1/n$.

6 Extensions

6.1 CBNs with Inputs (Conjunctive Boolean Control Networks)

A CBN with **inputs** is called a **conjunctive Boolean control network** (CBCN). The dynamics become:

$$X_i(k+1) = f_i(X_1(k), \ldots, X_n(k), U_1(k), \ldots, U_r(k))$$

where $U_1, \ldots, U_r$ are external control inputs.

Observability definition for CBCNs: Since the update functions are AND operators, the "most informative" control sequence is $U_i(k) = 1$ for all i and k. Any zero input can only obscure information (because AND with zero gives zero regardless of the state).

Definition 6.1 (Definition 3 from the paper)**.** A CBCN is said to be observable on $[0, N]$ if the CBN obtained by setting all inputs to one for all time $k \in [0, N-1]$ is observable on $[0, N]$.

In plain English: To test observability of a CBCN, set all inputs to one (which effectively removes them from the AND functions, since $\text{AND}(1, X_1, \ldots, X_k) = \text{AND}(X_1, \ldots, X_k)$), and then apply the theory developed for CBNs without inputs.

This means Algorithm 2 can be applied directly to CBCNs after this simplification.

6.2 CBNs with General AND Output Functions

The paper also considers CBNs where the output functions are AND operators (not just direct measurements of single SVs):

$$Y_i(k) = g_i(X_1(k), \ldots, X_n(k)), \quad \forall i \in [1, m], \tag{10}$$

with every g_i an AND operator.

The key insight is that this can be reduced to the simpler form (3). Consider an **augmented BN** with $n + m$ SVs:

$$\begin{aligned} \bar{X}_i(k+1) &= f_i(\bar{X}_1(k), \ldots, \bar{X}_n(k)), \quad \forall i \in [1, n], \\ \bar{X}_{n+j}(k+1) &= g_j(\bar{X}_1(k), \ldots, \bar{X}_n(k)), \quad \forall j \in [1, m], \\ \bar{Y}_p(k) &= \bar{X}_{n+p}(k), \quad \forall p \in [1, m]. \end{aligned} \tag{11}$$

In plain English: Add m new state variables (one per output) whose dynamics mirror the output functions. Then make these new variables the directly observable outputs. Since $\bar{X}_{n+j}(k+1) = g_j(\bar{X}_1(k), \ldots, \bar{X}_n(k)) = Y_j(k)$, this augmented system is observable if and only if the original system with AND outputs is observable.

The augmented system is still a CBN (all update functions are AND), and its outputs are direct measurements of SVs, so all the results of the paper apply.

7 Summary of Notation

For quick reference, here is a table of all major symbols used in the paper:

Symbol	Type / Range	Meaning
$S = \{0, 1\}$	Set	The Boolean domain
n	Integer ≥ 1	Number of state variables
m	Integer ≥ 0	Number of outputs (sensors)
k	Integer ≥ 0	Discrete time index
$[i, j]$	Set	$\{i, i+1, \ldots, j\}$
$X_i(k) \in S$	Binary	State variable i at time k
$X(k) \in S^n$	Binary vector	Full state at time k
$Y_j(k) \in S$	Binary	Output j at time k
$Y(k) \in S^m$	Binary vector	Full output at time k
$f_i : S^n \to S$	Function	Update (dynamics) function for SV i
$h_j : S^n \to S$	Function	Output function for output j
$\epsilon_{ji} \in \{0, 1\}$	Binary	1 iff X_j appears in f_i (CBN)
$G = (V, E)$	Directed graph	Dependency graph of the CBN
$e_{i \to j}$	Directed edge	Edge from X_i to X_j in G
$\mathcal{N}_{in}(X_i)$	Set of nodes	In-neighbors of X_i
$\mathcal{N}_{out}(X_i)$	Set of nodes	Out-neighbors of X_i
$\lvert\mathcal{N}_{in}(X_i)\rvert$	Integer	In-degree of X_i
Source	Node	Node with in-degree 0
Sink	Node	Node with out-degree 0
O_1	Property	Every hidden node is the sole in-neighbor of some node
O_2	Property	Every all-hidden cycle has an escape route
L_1	Set of nodes	Hidden nodes violating Property O_1
L_2	Set of nodes	Hidden nodes satisfying Property O_1
L_C	Set of cycles	Cycles in L_2 violating Property O_2
$\mathcal{I}$	Set of indices	Minimal set of SVs to make directly observable
L_{pairs}	$n \times 2$ matrix	Records "sole in-neighbor" pairs
p	$\in [0, 1]$	Edge probability in random graph
$q(p)$	$\in [0, 1]$	Probability a node has in-degree 1, no self-loop
$p^* = 1/n$	Probability	Optimal edge probability (maximizes q)
W	Set of nodes	Nodes with in-degree 1 and no self-loop
k	Integer	Minimal number of outputs for observability
$s = 100k/n$	Percentage	Fraction of nodes needing sensors
$\underline{s}(p)$	Percentage	Lower bound on s: $100(1 - q(p))$
$\bar{s}(p)$	Percentage	Upper bound on s: $100(1 - q(p)/2)$
$U_i(k) \in S$	Binary	Control input i at time k (CBCN)
$g_j : S^n \to S$	Function	AND output function (general case)
$\bar{X}_i$	Binary	State variable in augmented BN

8 Conceptual Summary: The Big Picture

Here is a bird's-eye view of the entire paper's logic:

1. **The problem:** Given a CBN (a system with AND-only updates), determine the smallest number of sensors to add so that the entire system state can be reconstructed from sensor measurements.
2. **The graph-theoretic reformulation:** Represent the CBN as a dependency graph. Observability reduces to two graph properties (O_1 and O_2) about in-degree-one nodes and cycles.
3. **The characterization (Theorem 1):** A CBN is observable $\iff$ Properties O_1 and O_2 hold $\iff$ the graph decomposes into disjoint observed paths.
4. **The algorithm (Algorithm 2, Theorem 2):** Find nodes violating O_1 (they must become sensors) and cycles violating O_2 (one sensor per cycle). This gives the minimal set. Running time: $O(n^2)$.
5. **Random graph analysis:** For random dependency graphs with edge probability p, the minimum fraction of nodes needing sensors is minimized at $p^* = 1/n$, where it is at least $\approx 63\%$. In practice, $\approx 70\%$ of nodes need sensors. The optimal density shows a sharp phase transition.
6. **Extensions:** The results extend to CBNs with inputs (set inputs to one) and CBNs with AND output functions (augment the state space).

Observability vs. Controllability

For CBNs, the minimal *controllability* problem (finding the fewest inputs to add so that the system can be driven to any desired state) is NP-hard. In contrast, the minimal *observability* problem is solvable in polynomial time ($O(n^2)$).
The fundamental reason for this asymmetry: adding an *output* (sensor) does not change the system dynamics or the dependency graph. Adding an *input* (control) changes the dynamics and thus changes the dependency graph, making the problem much harder.