Technical Companion to

Output Selection and Observer Design for Boolean Control Networks: A Sub-Optimal Polynomial-Complexity Algorithm

A Self-Contained Guide for CS/EE Undergraduates

Companion to the paper by Eyal Weiss and Michael Margaliot
IEEE Control Systems Letters, 2019

Abstract

This document explains every equation, definition, algorithm, and theoretical result in the paper "Output Selection and Observer Design for Boolean Control Networks: A Sub-Optimal Polynomial-Complexity Algorithm" in detail, assuming only undergraduate-level knowledge of discrete mathematics, Boolean logic, and basic graph theory. No prior knowledge of Boolean control networks, observability theory, or observer design is required. Each concept is accompanied by (1) a plain-English description, (2) a symbol-by-symbol breakdown, (3) an intuitive explanation of why the concept matters, and (4) concrete numerical examples where helpful.

Contents

1 Background: Boolean Networks, Control, and Observability

Before we can understand the paper's contributions, we need to build up several concepts from scratch: Boolean networks, Boolean *control* networks, observability, and observers. If you are comfortable with all of these, you may skip to Section 2.

1.1 What Is a Boolean Network?

A **Boolean network (BN)** is one of the simplest kinds of dynamical system. It is a discrete-time system where every variable takes values in $\{0, 1\}$ (equivalently, {False, True}) and updates according to a Boolean function.

Key Idea

Think of a Boolean network as a collection of light switches, each of which is either ON (1) or OFF (0). At every time step, each switch looks at the current states of some other switches and decides whether to turn ON or OFF at the next step, according to a fixed rule.

Formally, let $S := \{0, 1\}$ be the Boolean set. A BN with n **state variables (SVs)** $X_1, X_2, \ldots, X_n$ evolves as:

$$X_i(k+1) = f_i(X(k)), \qquad \forall\, i \in \{1, \ldots, n\},$$

where:

- $k = 0, 1, 2, \ldots$ is the discrete time index.
- $X(k) = [X_1(k), \ldots, X_n(k)]' \in S^n$ is the **state vector** at time k—an n-tuple of zeros and ones.
- $f_i : S^n \to S$ is a Boolean function that determines how state variable i updates. It can depend on any subset of the n state variables.

Numerical Example

Consider a BN with $n = 3$ state variables:

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

Here $\wedge$ is Boolean AND, $\vee$ is Boolean OR, and $\bar{\cdot}$ is Boolean NOT.
Starting from the initial state $X(0) = [1, 0, 1]$:

k	X_1	X_2	X_3
0	1	0	1
1	$0 \wedge 1 = 0$	$1 \vee 0 = 1$	$\overline{1} = 0$
2	$1 \wedge 0 = 0$	$0 \vee 1 = 1$	$\overline{0} = 1$
3	$1 \wedge 1 = 1$	$0 \vee 1 = 1$	$\overline{0} = 1$
4	$1 \wedge 1 = 1$	$1 \vee 1 = 1$	$\overline{1} = 0$

The system evolves deterministically once the initial state is fixed. Since there are only $2^3 = 8$ possible states, the trajectory must eventually repeat.

Why are BNs useful? BNs have been extensively used in systems biology, particularly for modeling **gene regulatory networks**. The state of each gene (expressed or not expressed) is modeled as a Boolean SV. The interactions between genes—e.g., the effect of the proteins they

encode on the promoter regions of other genes—determine the Boolean update function for each SV.

1.2 Boolean Control Networks (BCNs)

A **Boolean control network (BCN)** extends a BN by adding **control inputs**. These are external signals that an operator can choose at each time step to influence the system's behavior.

Definition 1.1 (Boolean Control Network)**.** A BCN with n SVs, p inputs, and m outputs is described by:

$$\begin{aligned} X_i(k+1) &= f_i(X(k), U(k)), \qquad \forall\, i \in [1, n], \\ Y_j(k) &= h_j(X(k)), \qquad \forall\, j \in [1, m], \end{aligned} \tag{Eq. 1}$$

where:

- $X(k) = [X_1(k), \ldots, X_n(k)]' \in S^n$ is the state vector.
- $U(k) = [U_1(k), \ldots, U_p(k)]' \in S^p$ is the input (control) vector.
- $Y(k) = [Y_1(k), \ldots, Y_m(k)]' \in S^m$ is the output vector.
- f_i and h_j are Boolean functions.

Symbol-by-symbol breakdown:

- $S := \{0, 1\}$: the Boolean set.
- $[1, n] := \{1, 2, \ldots, n\}$: a convenient shorthand for integer ranges.
- $f_i(X(k), U(k))$: the update function for state variable i. It takes the current state *and* the current control input and produces the next value of X_i.
- $h_j(X(k))$: the output function for output j. It maps the current state to a Boolean measurement. The output is what an external observer can actually *see*.
- The prime $'$ in $[X_1(k), \ldots, X_n(k)]'$ denotes a column vector (transpose of a row vector).

Inputs vs. Outputs: The Information Asymmetry

The key asymmetry in a BCN is:

- **Inputs** (U): chosen by the controller. We know what we applied.
- **States** (X): the internal "truth" of the system. We generally *cannot* measure all state variables.
- **Outputs** (Y): what we can actually measure. Each output is some function of the state, but there may be fewer outputs than states ($m < n$), so we only see a partial picture.

The central question of this paper is: *Which state variables should we measure (make into outputs) so that we can reconstruct the full internal state from the outputs and inputs?*

The paper further assumes that update functions only include arguments that actually influence the output. That is, if $X_i(k)$ is an argument of f_j, then there exists some assignment of the other variables such that changing X_i actually changes the value of f_j. Similarly for control inputs.

Numerical Example

Consider a BCN with $n = 3$ SVs, $p = 1$ input, and $m = 1$ output:

$$\begin{aligned} X_1(k+1) &= X_2(k) \wedge U_1(k), \\ X_2(k+1) &= X_3(k), \\ X_3(k+1) &= \overline{X_1(k)}, \\ Y_1(k) &= X_1(k). \end{aligned}$$

We can choose $U_1(k)$ at each time step, and we can observe $Y_1(k) = X_1(k)$. But we cannot directly observe X_2 or X_3. Can we still figure out the values of X_2 and X_3 from a time sequence of Y_1 and U_1? That is the observability question.

1.3 Observability and Reconstructability

Observability is a fundamental concept in control theory. Informally, a system is observable if you can determine its internal state from external measurements over time.

Definition 1.2 (Observability)**.** A BCN (Eq. 1) is **observable on** $[0, N]$ if for every control sequence $U := \{U(0), \ldots, U(N-1)\}$ and for any two different initial conditions $X(0)$ and $\tilde{X}(0)$, the corresponding output sequences $\{Y(0), \ldots, Y(N)\}$ and $\{\tilde{Y}(0), \ldots, \tilde{Y}(N)\}$ are different.

What this means in plain English: If two different starting states always produce different output sequences (no matter what inputs we apply), then by watching the outputs we can always tell which initial state the system started in. That is, the initial state is uniquely determined by the input–output record.

Symbol breakdown:

- $[0, N]$: the observation window—we watch the system for $N + 1$ time steps.
- "For every control sequence U": the definition requires distinguishability under *all* possible input sequences, not just some convenient one.
- $X(0) \neq \tilde{X}(0)$: two distinct initial states.
- $Y(0), \ldots, Y(N)$: the output sequence produced starting from $X(0)$ under input U.
- $\tilde{Y}(0), \ldots, \tilde{Y}(N)$: the output sequence produced starting from $\tilde{X}(0)$ under the same input U.
- The BCN is observable if these output sequences differ whenever $X(0) \neq \tilde{X}(0)$.

A BCN is called **observable** if it is observable on $[0, N]$ for some value $N \geq 0$.

Key Idea

Think of observability like this: you are in a dark room with a machine. You can press buttons (inputs) and read a small display (outputs). The machine has internal switches you cannot see (states). The system is observable if, by pressing buttons and reading the display over enough time steps, you can always figure out the exact position of every internal switch—no matter what those positions were initially.

Definition 1.3 (Reconstructability)**.** A BCN is **reconstructable** if for any integer $r > 0$, the knowledge of every admissible input and output trajectory $\{(Y(k), U(k))\}$, $k = 0, 1, \ldots, r$, uniquely determines the *final* state $X(r)$.

The difference is subtle: observability means you can determine the *initial* state; reconstructability means you can determine the *current* (final) state. If a BCN is observable, it is also

reconstructable—if you know the initial state, you can compute forward to find the current state.

Observability Is NP-Hard!

Testing whether a BCN is observable is NP-hard in the number of state variables. This means there is (likely) no efficient algorithm that can determine observability for all BCNs. This fact is a major motivation for the paper: since we cannot efficiently *check* observability, we look for *sufficient conditions*—conditions that guarantee observability, even if they are not necessary. A sufficient condition that can be checked in polynomial time is very valuable.

1.4 Directly Observable (Measurable) State Variables

Definition 1.4 (Directly Observable SV)**.** A state variable X_i is **directly observable** (or **directly measurable**) if there exists an output Y_j such that $Y_j(k) = X_i(k)$ for all k.

In other words, X_i is directly observable if one of the outputs is simply X_i itself—we can read its value directly from the output.

The paper focuses on the special case where every output is exactly equal to one SV:

$$Y_j(k) = X_j(k), \qquad j \in \{1, \ldots, m\}, \tag{Eq. 2}$$

assuming (without loss of generality) that the m outputs correspond to the first m SVs. Thus, nodes $X_1, \ldots, X_m$ are directly observable, and nodes $X_{m+1}, \ldots, X_n$ are *not* directly observable.

Numerical Example

If $n = 5$ and $m = 2$ with $Y_1(k) = X_1(k)$ and $Y_2(k) = X_2(k)$:

- X_1, X_2: directly observable (we can read their values).
- X_3, X_4, X_5: not directly observable (their values are hidden).

The challenge is to infer X_3, X_4, X_5 from the time sequences of X_1, X_2 and the known inputs.

1.5 What Is an Observer?

An **observer** (also called a **state estimator**) is an algorithm or system that takes the input and output sequences as input and produces an estimate of the full internal state at each time step.

In classical linear control theory, the famous **Luenberger observer** reconstructs the state of a linear system from its outputs. The BCN setting is fundamentally different because the dynamics are Boolean (nonlinear and discrete-valued), and standard linear techniques do not apply.

The paper constructs an observer called the **Disjoint-Path Observer**, whose computational complexity is polynomial in the size of the BCN description. This is significant because many previous BCN observers had *exponential* complexity.

2 The Dependency Graph

The paper's key insight is that the *structure* of the Boolean functions—specifically, which variables depend on which other variables—can be captured by a graph. Many properties needed for observability can be checked by analyzing this graph, *without* needing to examine the truth tables of the Boolean functions.

2.1 Directed Graphs: A Quick Refresher

A **directed graph (digraph)** $G = (V, E)$ consists of:

- A set of **vertices** V.
- A set of **directed edges (arcs)** $E \subseteq V \times V$. An edge $e_{i \to j}$ (or $(v_i \to v_j)$) denotes an arc from v_i to v_j.

Some terminology:

- v_i is an **in-neighbor** of v_j if there is an arc from v_i to v_j.
- v_j is an **out-neighbor** of v_i in the same case.
- $\mathcal{N}_{in}(v_i)$: the set of in-neighbors of v_i; $\mathcal{N}_{out}(v_i)$: the set of out-neighbors.
- **In-degree** of v_i: $|\mathcal{N}_{in}(v_i)|$; **out-degree**: $|\mathcal{N}_{out}(v_i)|$.
- A **source** has in-degree zero; a **sink** has out-degree zero.
- A **walk** from v_i to v_j is a sequence of vertices $v_{i_0}, v_{i_1}, \ldots, v_{i_q}$ with $v_{i_0} = v_i$, $v_{i_q} = v_j$, and $e_{i_k \to i_{k+1}} \in E$ for all k.
- A **cycle** is a closed walk where all vertices (except start = end) are distinct.

2.2 Defining the Dependency Graph

Definition 2.1 (Dependency Graph)**.** The **dependency graph** of a BCN is the directed graph $G = (V, E)$ where:

- $V = \{X_1, \ldots, X_n, U_1, \ldots, U_p\}$: one vertex for each SV and each input.
- $(X_i \to X_j) \in E$ if $X_i(k)$ is an argument of f_j—i.e., X_i appears in the update function of $X_j(k+1)$.
- $(U_q \to X_j) \in E$ if $U_q(k)$ is an argument of f_j.

What it captures: The dependency graph encodes the *actual variable dependencies* in the update functions. An edge from X_i to X_j means "the next value of X_j depends on the current value of X_i."

Important subsets:

- $V_s := \{X_1, \ldots, X_n\} \subseteq V$: the vertices corresponding to SVs only (no inputs).
- $E_s \subseteq E$: edges involving only SVs.
- $G_s := (V_s, E_s)$: the **reduced dependency graph**—the subgraph restricted to SVs.

The paper states that nodes $X_1, \ldots, X_m$ in the dependency graph correspond to directly observable SVs, and nodes $X_{m+1}, \ldots, X_n$ correspond to non-directly observable SVs.

Dependency Graph for the Paper's Example 1

Consider the BCN from the paper with $n = 5$ SVs, $p = 1$ input, and $m = 2$ outputs:

$$\begin{aligned}
X_1(k+1) &= X_3(k), \\
X_2(k+1) &= \overline{X_5(k)}, \\
X_3(k+1) &= \overline{X_4(k)}, \\
X_4(k+1) &= (X_2(k) \wedge X_3(k)) \vee U_4(k), \\
X_5(k+1) &= \overline{X_1(k)} \vee X_5(k), \\
Y_1(k) &= X_1(k), \quad Y_2(k) = X_2(k).
\end{aligned}$$

The dependency graph has vertices $\{X_1, X_2, X_3, X_4, X_5, U_4\}$ and directed edges:

Edge	Reason
$X_3 \to X_1$	$X_1(k+1) = X_3(k)$, so X_1 depends on X_3
$X_5 \to X_2$	$X_2(k+1) = \overline{X_5(k)}$
$X_4 \to X_3$	$X_3(k+1) = \overline{X_4(k)}$
$X_2 \to X_4$	$X_4(k+1)$ depends on $X_2(k)$
$X_3 \to X_4$	$X_4(k+1)$ depends on $X_3(k)$
$U_4 \to X_4$	$X_4(k+1)$ depends on $U_4(k)$
$X_1 \to X_5$	$X_5(k+1)$ depends on $X_1(k)$
$X_5 \to X_5$	$X_5(k+1)$ depends on $X_5(k)$ (self-loop)

The directly observable nodes are X_1 and X_2 (since $Y_1 = X_1$, $Y_2 = X_2$). The non-directly observable nodes are X_3, X_4, and X_5.
In the reduced graph G_s (SVs only), we remove vertex U_4 and the edge $U_4 \to X_4$.

3 The Sufficient Condition for Observability

The paper presents two graph-theoretic properties—P_1 and P_2—that together form a *sufficient condition* for observability.

3.1 Property P_1: Information Propagation

Definition 3.1 (Property P_1)**.** A BCN has Property P_1 if for every non-directly observable node X_i (i.e., $i > m$), there exists some other node X_j such that $\mathcal{N}_{in}(X_j) = \{X_i\}$.

What it says in plain English: Every hidden (non-directly-observable) SV X_i must have at least one "recipient" X_j whose update function depends on X_i *and only on* X_i—no other SVs and no inputs.

Symbol breakdown:

- $\mathcal{N}_{in}(X_j) = \{X_i\}$: the in-neighbor set of X_j in the *full* dependency graph G is exactly $\{X_i\}$. This means $X_j(k+1) = f_j(X_i(k))$—the update of X_j depends only on X_i, with no dependence on any other SV or any input.

Why this matters: When $\mathcal{N}_{in}(X_j) = \{X_i\}$, we have $X_j(k+1) = f_j(X_i(k))$. Since the update depends only on X_i (no other SVs, no inputs), and we assumed X_i genuinely affects f_j, it follows that f_j is either the identity function ($X_j(k+1) = X_i(k)$) or the NOT function ($X_j(k+1) = \overline{X_i(k)}$). In either case the value of $X_j(k+1)$ uniquely determines $X_i(k)$ and vice versa.

Property P_1: Each Hidden SV Has a "Relay"

Property P_1 says that the information on each hidden SV "propagates" to at least one other node. If $X_j(k+1)$ depends solely on $X_i(k)$ (no other SVs and no inputs), then knowing $X_j(k+1)$ tells us $X_i(k)$. The information about X_i is not "trapped"—it flows forward in time through X_j.
Imagine a chain of people whispering a message: P_1 ensures that every person whispers to at least one other person who is listening *only* to them, so the message can be perfectly recovered from the listener.

Numerical Example

Consider $X_3(k+1) = \overline{X_4(k)}$ from Example 1 in the paper. The in-neighbor set of X_3 in G is $\mathcal{N}_{in}(X_3) = \{X_4\}$ (no other SVs or inputs). So X_4 (a non-directly observable node) has X_3 as a relay: $\mathcal{N}_{in}(X_3) = \{X_4\}$. This means X_4 satisfies the P_1 requirement via X_3.
Similarly, $X_1(k+1) = X_3(k)$, so $\mathcal{N}_{in}(X_1) = \{X_3\}$. Node X_3 satisfies P_1 via X_1.
And $X_2(k+1) = \overline{X_5(k)}$, so $\mathcal{N}_{in}(X_2) = \{X_5\}$. Node X_5 satisfies P_1 via X_2.
So all three hidden SVs (X_3, X_4, X_5) satisfy Property P_1.

3.2 Property P_2: Breaking Cycles

Definition 3.2 (Property P_2)**.** A BCN has Property P_2 if every cycle C in its dependency graph that is composed solely of non-directly observable nodes satisfies the following property: 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.

What it says in plain English: If there is a cycle of hidden SVs, then at least one SV in the cycle must "leak" its information out of the cycle—to a node X_j outside the cycle, and X_j must depend *only* on that SV (no other SVs and no inputs).

Why this matters: A cycle among hidden SVs is dangerous because the information might circulate endlessly within the cycle without ever reaching an observable node. Property P_2 ensures that every such cycle has an "exit ramp": the state of at least one node in the cycle can be determined from a node outside the cycle.

Property P_2: Every Hidden Cycle Has an Exit

Property P_2 prevents information from being "trapped" in cycles. Think of a circular conveyor belt: items go around and around. P_2 requires that at least one position on the belt has a chute that diverts a copy of the item off the belt, where it can be inspected.

Numerical Example

Suppose the reduced dependency graph G_s has a cycle $X_3 \to X_4 \to X_3$ (both non-directly observable). For P_2 to hold, there must exist a node X_j outside this cycle such that $\mathcal{N}_{in}(X_j) = \{X_3\}$ or $\mathcal{N}_{in}(X_j) = \{X_4\}$.
If $X_1(k+1) = X_3(k)$ and X_1 is directly observable and not in the cycle, then $\mathcal{N}_{in}(X_1) = \{X_3\}$ and $X_1 \notin C$. So P_2 is satisfied: the cycle has an exit through $X_3 \to X_1$.
If, instead, every node outside the cycle had in-degree ≥ 2 or depended on SVs other than X_3 or X_4, then P_2 would fail.

3.3 The Main Theorem: $P_1 + P_2 \implies$ Observable

Theorem 3.1 (Theorem 1 in the paper)**.** *A BCN that satisfies Properties P_1 and P_2 is observable.*

What it says in plain English: If every hidden SV has a dedicated relay (Property P_1) and every cycle of hidden SVs has an exit (Property P_2), then the full state of the BCN can be determined from the input–output record.

Why is this only a *sufficient* condition? The condition is sufficient but *not necessary*. This means:

- If P_1 and P_2 hold, the BCN is *definitely* observable.

- If P_1 or P_2 fails, the BCN *might* still be observable for other reasons (but our algorithm cannot certify this).

This trade-off is deliberate: checking the exact (necessary and sufficient) condition for observability is NP-hard, but checking P_1 and P_2 can be done in *polynomial time*. We sacrifice some precision for computational tractability.

Sufficient $\neq$ Necessary

A BCN that violates P_1 or P_2 is not necessarily unobservable—it just means this particular test cannot confirm observability. Think of it like a fire alarm: if the alarm goes off, there is a fire (sufficient); but the alarm might not detect a small fire (not necessary). The condition might add more sensors than strictly necessary, but it guarantees observability.

4 Observed Paths: The Structural Key

The proof of Theorem 1 relies on a structural concept called **observed paths**. These paths form the backbone of the observer construction.

4.1 What Is an Observed Path?

Definition 4.1 (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 has $p > 1$ elements, then for every element $i < p$:
   - Element i is a non-directly observable node.
   - Element i is the *only* element in the in-neighbors set of element $i + 1$.

What it means in plain English: An observed path is a chain of SVs, ending at a directly observable SV, where each SV in the chain is the *sole input* (no other SVs, no control inputs) to the next SV in the chain. The information flows along the chain like dominoes, ultimately reaching an output.

Observed Path $\approx$ Shift Register

An observed path corresponds to a "shift register"—a chain of flip-flops where each flip-flop passes its value to the next one at each clock tick. The last flip-flop is connected to a display (output), so by reading the display over successive time steps, you see the value that each flip-flop had in the past.

Numerical Example

In the paper's Example 1, one observed path is (X_4, X_3, X_1):
- X_1 is directly observable (it is the last element). ✓
- X_3 is non-directly observable. $\mathcal{N}_{in}(X_1) = \{X_3\}$—$X_3$ is the only in-neighbor of X_1. ✓
- X_4 is non-directly observable. $\mathcal{N}_{in}(X_3) = \{X_4\}$—$X_4$ is the only in-neighbor of X_3. ✓

The information flows: $X_4 \to X_3 \to X_1$. Since $X_1(k+1) = X_3(k)$ and $X_3(k+1) = \overline{X_4(k)}$,

we have:

$$Y_1(0) = X_1(0),$$
$$Y_1(1) = X_1(1) = X_3(0),$$
$$Y_1(2) = X_1(2) = X_3(1) = \overline{X_4(0)}.$$

By reading Y_1 at times 0, 1, and 2, we recover $X_1(0) = Y_1(0)$, $X_3(0) = Y_1(1)$, and $X_4(0) = \overline{Y_1(2)}$ (inverting because f_3 is the NOT function). The observed path of length 3 reveals all three SVs' initial values over 3 time steps!

4.2 Disjoint Observed Paths

Observed paths with non-overlapping nodes are called **disjoint observed paths**.

Proposition 4.1 (Proposition 1 in the paper)**.** *Consider a BCN that satisfies Properties P_1 and P_2. Then:*

1. *G_s can be decomposed into disjoint observed paths, such that every vertex in G_s belongs to exactly one observed path.*
2. *For every vertex $v \in (V \setminus V_s)$ (i.e., input vertices), $\mathcal{N}_{out}(v)$ contains only vertices that are at the* beginning *of observed paths.*

What this means: Under conditions P_1 and P_2, the entire reduced dependency graph can be partitioned into chains that each end at an output. Every SV belongs to exactly one chain. Inputs feed into the beginnings of these chains only.

Numerical Example

For the paper's Example 1, Algorithm 1 (described below) produces two disjoint observed paths:

$$O^1 = (X_4, X_3, X_1),$$
$$O^2 = (X_5, X_2).$$

Every SV belongs to exactly one path:
- $X_4 \in O^1$, $X_3 \in O^1$, $X_1 \in O^1$.
- $X_5 \in O^2$, $X_2 \in O^2$.

The input U_4 feeds into X_4, which is at the beginning of O^1—consistent with the proposition.

5 Algorithm 1: Decomposing into Disjoint Observed Paths

Algorithm 1 takes the dependency graph and decomposes the SV-nodes of G_s into disjoint observed paths.

5.1 High-Level Idea

The algorithm works as follows:

1. For each directly observable node X_i (from $i = 1$ to m), start a new path with X_i at the end.

2. Trace backwards: if X_i has exactly one in-neighbor v in G_s that is a SV, and v has not been assigned to another path yet, and v is not directly observable, then prepend v to the path. Repeat from v.
3. When the tracing cannot continue (either in-degree $\neq 1$, or the in-neighbor is already in a path, or the in-neighbor is directly observable), output the current path.

5.2 Step-by-Step Pseudocode

Algorithm 1 Decompose the nodes of G_s into disjoint observed paths

Require: Dependency graph G of a BCN satisfying P_1 and P_2
Ensure: A decomposition of G_s into m disjoint observed paths

```
1:  for i = 1 to m do
2:    o-node ← X_i {start from directly observable node X_i}
3:    o-path ← {X_i}
4:    if |N_in(o-node)| = 1 then
5:      Let {v} = N_in(o-node)
6:      if v does not belong to a previous path, v ∈ V_s, and v is not directly observable then
7:        Insert v at the front of o-path, just before o-node
8:        o-node ← v; go to line 4
9:      end if
10:   end if
11:   Print o-path
12: end for
```

Line-by-line explanation:

- **Line 1**: Iterate over all m directly observable nodes. Each iteration builds one observed path.
- **Lines 2–3**: Initialize the path with just the directly observable node. The variable "o-node" tracks the current "head" of the path (the node we are trying to extend backwards).
- **Line 4**: Check if o-node has exactly one in-neighbor in the full graph G (including both SVs and inputs). If it has zero or more than one, we cannot extend the path (the chain ends here).
- **Line 5**: Extract the unique in-neighbor v.
- **Line 6**: Two additional conditions must hold to extend the path:
  1. v is not already in another observed path (no overlap).
  2. v is an SV node (not an input) and is not directly observable.

  (Note: If $|\mathcal{N}_{in}(\text{o-node})| = 1$ and $v \in V_s$, then v is necessarily the sole in-neighbor in G, which is exactly the condition used in Properties P_1 and P_2.)
- **Lines 7–8**: If all conditions hold, prepend v to the path and continue tracing from v.
- **Lines 9–10**: If we cannot extend, print the completed path.

Tracing Algorithm 1 on Example 1

The BCN has $m = 2$ directly observable nodes: X_1 and X_2.
Iteration $i = 1$ (starting from X_1):
1. o-path $= \{X_1\}$, o-node $= X_1$.

2. $\mathcal{N}_{in}(X_1) = \{X_3\}$, so $|\mathcal{N}_{in}| = 1$. Check: X_3 not in any path, $X_3 \in V_s$, X_3 not directly observable. All pass.
3. Prepend: o-path $= \{X_3, X_1\}$, o-node $= X_3$.
4. $\mathcal{N}_{in}(X_3) = \{X_4\}$, so $|\mathcal{N}_{in}| = 1$. Check: X_4 not in any path, $X_4 \in V_s$, X_4 not directly observable. All pass.
5. Prepend: o-path $= \{X_4, X_3, X_1\}$, o-node $= X_4$.
6. $\mathcal{N}_{in}(X_4) = \{X_2, X_3, U_4\}$ in G, so $|\mathcal{N}_{in}| = 3 \neq 1$. Stop.
7. Print: $O^1 = (X_4, X_3, X_1)$.

Iteration $i = 2$ (starting from X_2):

1. o-path $= \{X_2\}$, o-node $= X_2$.
2. $\mathcal{N}_{in}(X_2) = \{X_5\}$ in G, so $|\mathcal{N}_{in}| = 1$. Check: X_5 not in any path, $X_5 \in V_s$, X_5 not directly observable. All pass.
3. Prepend: o-path $= \{X_5, X_2\}$, o-node $= X_5$.
4. $\mathcal{N}_{in}(X_5) = \{X_1, X_5\}$ in G, so $|\mathcal{N}_{in}| = 2 \neq 1$. Stop.
5. Print: $O^2 = (X_5, X_2)$.

Result: G_s is decomposed into two disjoint observed paths: $O^1 = (X_4, X_3, X_1)$ and $O^2 = (X_5, X_2)$. Every SV belongs to exactly one path.

5.3 Correctness of Algorithm 1

The paper proves that Algorithm 1 correctly decomposes G_s into disjoint observed paths whenever the BCN satisfies P_1 and P_2. The key notation used in the proof is $v_p \mapsto v_q$, meaning "v_p points to v_q" in the sense that:

- There is an edge from v_p to v_q in the dependency graph, and
- $\mathcal{N}_{in}(v_q) = \{v_p\}$ (i.e., v_p is the *only* in-neighbor of v_q).

Claim: The algorithm outputs an observed path that contains every non-directly-observable node X_j.

Proof sketch: By P_1, there exists $k \neq j$ such that $X_j \mapsto X_k$.

- If X_k is directly observable ($k \leq m$), the algorithm traces back from X_k and includes X_j.
- If X_k is not directly observable, we apply P_1 again to X_k, finding X_h with $X_k \mapsto X_h$, and so on.
- If this chain ever enters a cycle of non-directly-observable nodes, P_2 guarantees there is an exit from the cycle to a node outside it, which eventually reaches a directly observable node.
- The algorithm traces back along this chain and includes X_j in the resulting path.

The disjointness follows because each node is checked against the "already assigned" condition (line 6 of the algorithm).

6 Observer Design: The Disjoint-Path Observer

Once the dependency graph is decomposed into disjoint observed paths, we can build an observer.

6.1 Corollary 1: The Observer Construction

Corollary 6.1 (Corollary 1 in the paper)**.** *Consider a BCN that satisfies the sufficient condition stated in Theorem 1. An observer for this BCN can be designed as follows:*

(a) Construct the dependency graph G.

(b) Apply Algorithm 1 to decompose G_s into m disjoint observed paths.

(c) Observe an output sequence of length equal to the longest observed path.

(d) Map the values observed at each output to the values of the SVs composing the observed paths, to obtain the initial state $X(0)$.

What this means: Once we know the initial state $X(0)$, we can compute $X(k)$ for all future k using the known dynamics and the known input sequence.

6.2 How the Observer Works: Detailed Mechanics

Consider an observed path $O^i = (X_{i_1}, X_{i_2}, \ldots, X_{i_{N_i}})$ of length N_i. The last element $X_{i_{N_i}}$ is directly observable.

The output of this path at times $0, 1, \ldots, N_i - 1$ reveals:

$$\begin{aligned} X_{i_{N_i}}(0) &= X_{i_{N_i}}(0), \\ X_{i_{N_i}}(1) &= f_{i_{N_i}}(X_{i_{N_i-1}}(0)), \\ X_{i_{N_i}}(2) &= f_{i_{N_i}}(f_{i_{N_i-1}}(X_{i_{N_i-2}}(0))), \\ &\vdots \\ X_{i_{N_i}}(N_i - 1) &= f_{i_{N_i}}(f_{i_{N_i-1}}(\ldots f_{i_2}(X_{i_1}(0))\ldots)). \end{aligned}$$

Each function f_{i_k} in the chain depends on only one SV (by the observed path property), so it is either the identity or the NOT function. Therefore each f_{i_k} is invertible, and its inverse is itself: $f_{i_k}^{-1} = f_{i_k}$.

This means we can recover the initial values backwards:

$$\begin{aligned} X_{i_{N_i}}(0) &= Y_{\text{output}}(0), \\ X_{i_{N_i-1}}(0) &= f_{i_{N_i}}^{-1}(X_{i_{N_i}}(1)) = f_{i_{N_i}}(Y_{\text{output}}(1)), \\ X_{i_{N_i-2}}(0) &= f_{i_{N_i-1}}^{-1}(f_{i_{N_i}}^{-1}(X_{i_{N_i}}(2))), \\ &\vdots \end{aligned}$$

Observer Reconstruction for Example 1

Observed path $O^1 = (X_4, X_3, X_1)$ with $Y_1 = X_1$.
The dynamics along this path are:

- $X_1(k+1) = X_3(k)$, so f_1 is the identity: $f_1(X_3) = X_3$.
- $X_3(k+1) = \overline{X_4(k)}$, so f_3 is the NOT function: $f_3(X_4) = \overline{X_4}$.

The output at successive times gives:

$$\begin{aligned} Y_1(0) &= X_1(0), \\ Y_1(1) &= X_1(1) = X_3(0), \\ Y_1(2) &= X_1(2) = X_3(1) = \overline{X_4(0)}. \end{aligned}$$

So if we observe $Y_1(0) = 1$, $Y_1(1) = 0$, $Y_1(2) = 0$:

- $X_1(0) = Y_1(0) = 1$.
- $X_3(0) = Y_1(1) = 0$.
- $X_4(0) = \overline{Y_1(2)} = \overline{0} = 1$ (since f_3 is NOT, we invert).

Similarly, observed path $O^2 = (X_5, X_2)$ with $Y_2 = X_2$:

- $X_2(k+1) = \overline{X_5(k)}$, so f_2 is the NOT function.

If we observe $Y_2(0) = 0$, $Y_2(1) = 0$:

- $X_2(0) = Y_2(0) = 0$.
- $X_5(0) = \overline{Y_2(1)} = \overline{0} = 1$ (since f_2 is NOT, we invert).

The full initial state is $X(0) = [1, 0, 0, 1, 1]$. The observation window needed is $\max(3, 2) - 1 = 2$ time steps (i.e., observe on $[0, 2]$). This works for *every* control sequence—we did not need to know U at all to reconstruct $X(0)$!

The Functions Along the Path Must Be Invertible

The observer works because each function in the chain is either identity or NOT, both of which are invertible. This is guaranteed by the structure of the observed path: when the in-neighbor set has exactly one element (a single SV, no inputs), the Boolean function of one variable is either $f(x) = x$ or $f(x) = \bar{x}$. Both are their own inverses. If a function depended on multiple SVs, it would generally not be invertible (e.g., AND of two variables cannot be inverted: $0 \wedge 0 = 0$ and $1 \wedge 0 = 0$).

7 Algorithm 2: Output Selection for Minimal Observability

So far, we have assumed that the outputs are given and we check whether the BCN is observable. But what if we get to *choose* which SVs to observe?

7.1 The Minimal Observability Problem

Problem 1. Given a BCN with n SVs, determine a minimal set of indices $\mathcal{I} \subseteq [1, n]$ such that making each X_i, $i \in \mathcal{I}$, directly measurable yields an observable BCN.

In plain English: We want to place as few sensors as possible, where each sensor directly reads the value of one SV, so that the resulting BCN is observable.

This problem is NP-hard in general (because it requires checking observability, which is NP-hard). Algorithm 2 provides a **sub-optimal** but polynomial-time solution: it finds a set of SVs that, when made directly observable, guarantees observability via the sufficient condition (Theorem 1), but this set may not be the absolute minimum.

7.2 The Idea Behind Algorithm 2

Algorithm 2 is based on an algorithm from reference [33] in the paper (which solved the same problem for the special case of conjunctive BCNs, where all update functions use only AND gates). The general idea is:

1. Create three lists:
   - L_1: SVs that are not directly observable and are *not* the only element in the in-neighbors set of any other node. (These violate P_1—no other node depends solely on them.)
   - L_2: SVs that are not directly observable and *are* the only element in the in-neighbors set of some node. (These satisfy P_1.)
   - L_C: cycles composed solely of nodes in L_2.
2. For each cycle $C \in L_C$, check if one of its elements appears as the only in-neighbor of a node not in C. If so, P_2 is satisfied for C, and remove C from L_C.

3. Return the SVs in L_1 plus one element from each remaining cycle in L_C. Making these SVs directly observable ensures that P_1 and P_2 hold.

7.3 Why This Works

- Nodes in L_1 violate P_1 because no other node depends solely on them. Making them directly observable eliminates the need for them to satisfy P_1 (directly observable nodes are excluded from the P_1 requirement).
- Nodes in L_2 already satisfy P_1, so they do not need to be made observable.
- Cycles in L_C that are not resolved by P_2 can be broken by making one node in each cycle directly observable—this removes that node from the "non-directly observable" category, breaking the cycle of hidden nodes.

7.4 Algorithm 2: Formal Description

Algorithm 2 Output selection for meeting the condition of Theorem 1 (high-level)

Require: A BCN (Eq. 2) with n SVs and $m \geq 0$ existing outputs
Ensure: A set of SVs so that making them directly observable yields a BCN satisfying P_1 and P_2

Construct the dependency graph G
Compute L_1: non-directly observable SVs X_i with $\mathcal{N}_{in}(X_j) \neq \{X_i\}$ for all j
Compute L_2: non-directly observable SVs X_i such that $\mathcal{N}_{in}(X_j) = \{X_i\}$ for some j
Compute L_C: cycles in G_s composed solely of nodes in L_2
for each cycle $C \in L_C$ **do**
  if some node in C has $\mathcal{N}_{in}(X_j) = \{X_i\}$ with $X_j \notin C$ **then**
    Remove C from L_C {P_2 is already satisfied for C}
  end if
end for
return $L_1 \cup$ {one node from each remaining $C \in L_C$}

Theorem 7.1 (Theorem 2 in the paper)**.** *Algorithm 2 provides a solution that satisfies the conditions of Theorem 1.*

Corollary 7.2 (Corollary 2 in the paper)**.** *An upper bound for the minimal size of the solution to Problem 1 is given by the size of the solution generated by Algorithm 2.*

Sub-Optimal but Polynomial

Algorithm 2 may select *more* outputs than the true minimum. However, it runs in polynomial time, whereas finding the exact minimum is NP-hard. For the special case of conjunctive BCNs (where all update functions are AND gates), Algorithm 2 becomes *optimal* (finds the true minimum).

Output Selection for the Mammalian Cell Cycle

The paper applies Algorithm 2 to a BCN model of the mammalian cell cycle with $n = 9$ SVs and $p = 1$ input. The input is constant (either always True or always False), so it acts like a parameter.

Under the constant input assumption, the BCN dynamics are:

$$x_1(t+1) = (\bar{u}(t) \wedge \bar{x}_3(t) \wedge \bar{x}_4(t) \wedge \bar{x}_9(t)) \vee (x_5(t) \wedge \bar{u}(t) \wedge \bar{x}_9(t)),$$
$$x_2(t+1) = (\bar{x}_1(t) \wedge \bar{x}_4(t) \wedge \bar{x}_9(t)) \vee (x_5(t) \wedge \bar{x}_1(t) \wedge \bar{x}_1(t) \wedge \bar{x}_9(t)),$$
$$\vdots \quad \text{(9 update functions in total)}$$

Algorithm 2 generates:

- $L_1 = \{X_1, \ldots, X_8\}$ (8 SVs do not appear as the sole in-neighbor of any other node).
- $L_2 = \{X_9\}$ (only X_9 is the sole in-neighbor of some other node).
- $L_C = \emptyset$ (no cycles composed solely of L_2 nodes).

The output: make $X_1, \ldots, X_8$ directly observable. This leaves only X_9 as non-directly-observable, and X_9 satisfies P_1 (it is in L_2), and there are no problematic cycles.
In fact, using the STP (semi-tensor product) representation for verification, the paper confirms that the minimum number of outputs for observability is 8, so Algorithm 2 achieves the *optimal* solution in this case. Note: the STP approach is feasible here only because $n = 9$ is small (the transition matrix is already $2^9 \times 2^9 = 512 \times 512$). For $n \geq 30$, STP-based methods become infeasible.

8 Complexity Analysis

One of the paper's main selling points is that both algorithms run in **polynomial time**, in contrast to most existing approaches that require **exponential time**.

8.1 What Is the "Description Length" of a BCN?

The complexity is measured in terms of the **length of the description** of the BCN. This is the total size of the Boolean expressions defining all update functions. It is closely related to $|V| + |E|$ in the dependency graph.

The resulting dependency graph has:

- $|V| = n + p$ vertices (one per SV, one per input).
- $|E| \leq n^2 + pn$ edges (each SV can depend on at most n SVs and p inputs).

8.2 Complexity of Algorithm 1 (Disjoint Path Decomposition)

Step	**Complexity**
Build dependency graph G	$O(\|V\| + \|E\|)$—linear in description length
Algorithm 1 main loop	$O(n)$: visits each vertex of V_s at most once, performing $O(1)$ operations per vertex
Total for decomposition	$O(\|V\| + \|E\|) = O(n + p + n^2 + pn) = O(n^2 + pn)$

Why Algorithm 1 is $O(n)$ (after graph construction): The outer loop runs m times. The inner loop extends the path by visiting new nodes. Each node is visited at most once (because nodes are not shared between paths). The total number of inner loop iterations across all m outer iterations is at most n. Each iteration performs $O(1)$ work (checking in-degree, looking up whether a node is already assigned).

8.3 Complexity of the Observer

Step	Complexity
Determine $X(0)$ from output sequence	$O(n)$—process the output values along each path
Compute $X(k)$ for each subsequent time step	$O(\|V\|+\|E\|)$ per step—evaluate all update functions once
Total for T time steps	$O(n+T\cdot(\|V\|+\|E\|))$

The observation window needed is $\max_{i=1,\ldots,m}\{N_i\}-1$ time steps, where N_i is the length of the i-th observed path. In the worst case, this is $n-m+1$ (one long path).

8.4 Complexity of Algorithm 2 (Output Selection)

The complexity analysis of Algorithm 2 is done in reference [33] of the paper, yielding a runtime that is **linear in the description length**: $O(|V|+|E|)$.

8.5 Comparison with Existing Methods

Method	Complexity	Exact?
STP-based approaches	$O(2^n)$ (exponential)	Yes (necessary & sufficient)
Shift-Register observer	$O(2^n)$ (exponential)	Yes
Multiple States observer	$O(2^n)$ (exponential)	Yes
Luenberger-like observer	$O(2^n)$ (exponential)	Yes
Disjoint-Path Observer (this paper)	$O(n^2+pn)$ (polynomial)	No (sufficient only)

The Polynomial vs. Exponential Trade-off

All previous methods were *exact*—they used necessary and sufficient conditions. But their $O(2^n)$ complexity makes them unusable for networks with, say, $n \geq 30$ SVs.
The approach in this paper is *approximate*—it uses a sufficient condition that may add extra outputs. But its polynomial complexity makes it practical for networks with hundreds or thousands of SVs.
In systems biology, gene regulatory networks easily have $n = 100$ or more genes. $2^{100} \approx 10^{30}$ operations are completely infeasible, but $100^2 + 100 = 10{,}100$ operations take microseconds.

Scaling: How Big a Difference Does This Make?

n	STP-based (2^n)	Disjoint-Path (n^2)
10	1,024	100
20	1,048,576	400
30	$\sim 10^9$	900
50	$\sim 10^{15}$	2,500
100	$\sim 10^{30}$	10,000
1,000	$\sim 10^{301}$	1,000,000

At $n = 30$, the STP approach already requires about a billion operations. At $n = 100$, it requires more operations than atoms in the observable universe. The polynomial algorithm remains comfortably tractable at any practical scale.

9 The Conjunctive BCN Special Case

9.1 What Are Conjunctive BCNs?

A **conjunctive BCN (CBCN)** is a special case where every update function uses only AND ($\wedge$) operations:

$$X_i(k+1) = \bigwedge_{j \in \text{args}(f_i)} X_j(k).$$

(Recall that $\wedge$ is Boolean AND: the output is 1 only if *all* inputs are 1.)

Conjunctive BCNs arise naturally in gene regulation: a gene is expressed only if *all* of its activators are present and *none* of its repressors are active.

9.2 Why CBCNs Are Special

For CBCNs, a necessary and sufficient condition for observability was derived in reference [33]. This condition is exactly Properties P_1 and P_2. In other words, for CBCNs, the sufficient condition in this paper becomes *also necessary*—it is tight.

This means that for CBCNs:

- Algorithm 2 finds the *minimum* number of outputs for observability.
- The Disjoint-Path Observer is optimal.

For general BCNs (with OR, NOT, and other operations), P_1 and P_2 remain only sufficient, and the algorithms may add more outputs than necessary.

10 Proof of Theorem 1: How P_1 and P_2 Guarantee Observability

This section explains the proof of the main theorem at a high level.

10.1 The Proof Strategy

The proof shows the following chain of implications:

$$(a) \implies (b) \implies (c)$$

where:

(a) The dependency graph G has Properties P_1 and P_2.
(b) There exists a decomposition of the dependency graph into $m \geq 1$ disjoint observed paths $O^1, \ldots, O^m$, such that every vertex in G_s belongs to a single observed path.
(c) The BCN is observable.

10.2 $(a) \implies (b)$: Algorithm 1 Produces the Decomposition

This is established by the correctness proof of Algorithm 1 (Section 5.3). Properties P_1 and P_2 ensure that every node can be traced back to a directly observable node via a chain of sole-dependency relationships.

10.3 $(b) \implies (c)$: The Decomposition Implies Observability

Suppose the decomposition exists with observed paths $O^i = (X_{i_1}, \ldots, X_{i_{N_i}})$ for $i = 1, \ldots, m$.

The output of path O^i at successive times is:

$$\begin{aligned}
X_{i_{N_i}}(0) &= X_{i_{N_i}}(0), \\
X_{i_{N_i}}(1) &= f_{i_{N_i}}(X_{i_{N_i-1}}(0)), \\
X_{i_{N_i}}(2) &= f_{i_{N_i}}(f_{i_{N_i-1}}(X_{i_{N_i-2}}(0))), \\
&\vdots \\
X_{i_{N_i}}(N_i - 1) &= f_{i_{N_i}}(f_{i_{N_i-1}}(\ldots f_{i_2}(X_{i_1}(0))\ldots)).
\end{aligned}$$

Each f_{i_k} is a function of a single SV, so it is either identity or NOT. Both are invertible (and self-inverse): $f_{i_k}^{-1} = f_{i_k}$.

Therefore, we can recover:

$$\begin{aligned}
X_{i_{N_i-1}}(0) &= f_{i_{N_i}}(X_{i_{N_i}}(1)), \\
X_{i_{N_i-2}}(0) &= f_{i_{N_i-1}}(f_{i_{N_i}}(X_{i_{N_i}}(2))), \\
&\vdots \\
X_{i_1}(0) &= f_{i_2}(f_{i_3}(\ldots f_{i_{N_i}}(X_{i_{N_i}}(N_i - 1))\ldots)).
\end{aligned}$$

Since every SV belongs to exactly one path, all initial values can be recovered. The observation window needed is $[0, \max_i\{N_i\} - 1]$, and this works for *every* control sequence (the recovery depends only on the output values and the functions in the chain, not on the inputs).

11 Putting It All Together: Complete Worked Example

Let us walk through the entire pipeline on the paper's Example 1.

11.1 Step 1: The BCN

$$\begin{aligned}
X_1(k+1) &= X_3(k), \\
X_2(k+1) &= \overline{X_5(k)}, \\
X_3(k+1) &= \overline{X_4(k)}, \\
X_4(k+1) &= (X_2(k) \wedge X_3(k)) \vee U_4(k), \\
X_5(k+1) &= \overline{X_1(k)} \vee X_5(k), \\
Y_1(k) &= X_1(k), \quad Y_2(k) = X_2(k).
\end{aligned}$$

Parameters: $n = 5$, $p = 1$, $m = 2$.

11.2 Step 2: Build the Dependency Graph

Vertices: $\{X_1, X_2, X_3, X_4, X_5, U_4\}$.

Edges (from the update functions): $X_3 \to X_1$, $X_5 \to X_2$, $X_4 \to X_3$, $X_2 \to X_4$, $X_3 \to X_4$, $U_4 \to X_4$, $X_1 \to X_5$, $X_5 \to X_5$.

Reduced graph G_s (SVs only, remove U_4): $X_3 \to X_1$, $X_5 \to X_2$, $X_4 \to X_3$, $X_2 \to X_4$, $X_3 \to X_4$, $X_1 \to X_5$, $X_5 \to X_5$.

11.3 Step 3: Verify Properties P_1 and P_2

In-neighbor sets in G (full dependency graph):

Node	$\mathcal{N}_{in}$ in G
X_1	$\{X_3\}$
X_2	$\{X_5\}$
X_3	$\{X_4\}$
X_4	$\{X_2, X_3, U_4\}$
X_5	$\{X_1, X_5\}$

Property P_1: Non-directly observable nodes are X_3, X_4, X_5.

- X_3: $\mathcal{N}_{in}(X_1) = \{X_3\}$ — X_3 is the sole in-neighbor of X_1. ✓
- X_4: $\mathcal{N}_{in}(X_3) = \{X_4\}$ — X_4 is the sole in-neighbor of X_3. ✓
- X_5: $\mathcal{N}_{in}(X_2) = \{X_5\}$ — X_5 is the sole in-neighbor of X_2. ✓

P_1 holds.

Property P_2: Are there cycles composed solely of non-directly observable nodes? The non-directly observable nodes are $\{X_3, X_4, X_5\}$.

- X_5 has a self-loop ($X_5 \to X_5$), which is a cycle of length 1. For P_2: X_5 must have $\mathcal{N}_{in}(X_j) = \{X_5\}$ for some X_j not in the cycle. Indeed, $\mathcal{N}_{in}(X_2) = \{X_5\}$ and $X_2 \notin \{X_5\}$. ✓
- There is also a cycle $X_3 \to X_4 \to X_3$ (since $X_4 \to X_3$ and $X_3 \to X_4$ are both edges). For P_2: the cycle must include a node X_i with $\mathcal{N}_{in}(X_j) = \{X_i\}$ for some X_j not in the cycle. Indeed, X_3 satisfies this: $\mathcal{N}_{in}(X_1) = \{X_3\}$ in G, and X_1 is directly observable and not in the cycle $\{X_3, X_4\}$. ✓

P_2 holds. By Theorem 1, the BCN is observable.

11.4 Step 4: Run Algorithm 1

As computed in Section 5: $O^1 = (X_4, X_3, X_1)$, $O^2 = (X_5, X_2)$.

11.5 Step 5: Build and Run the Observer

Suppose $X(0) = [1, 0, 0, 1, 1]$ and $U_4 \equiv 0$.

Compute the trajectories:

k	X_1	X_2	X_3	X_4	X_5	U_4
0	1	0	0	1	1	0
1	0	0	0	0	1	0
2	0	0	1	0	1	0

Verification: at $k = 0 \to 1$:

- $X_1(1) = \overline{X_3(0)} = \overline{0} = 0$. ✓
- $X_2(1) = \overline{X_5(0)} = \overline{1} = 0$. ✓
- $X_3(1) = \overline{X_4(0)} = \overline{1} = 0$. ✓
- $X_4(1) = (X_2(0) \wedge X_3(0)) \vee U_4(0) = (0 \wedge 0) \vee 0 = 0$. ✓
- $X_5(1) = \overline{X_1(0)} \vee X_5(0) = \overline{1} \vee 1 = 0 \vee 1 = 1$. ✓

The observer sees: $Y_1(0) = 1$, $Y_1(1) = 0$, $Y_1(2) = 0$, $Y_2(0) = 0$, $Y_2(1) = 0$.

From $O^1 = (X_4, X_3, X_1)$:

- $X_1(0) = Y_1(0) = 1$.
- $X_3(0) = \overline{Y_1(1)} = 0$ (since $X_1(k+1) = X_3(k)$ and $f_1 = \text{identity}$).
- $X_4(0) = \overline{Y_1(2)} = \overline{0} = 1$ (since $X_3(k+1) = \overline{X_4(k)}$ and $f_3 = \text{NOT}$, we invert).

From $O^2 = (X_5, X_2)$:

- $X_2(0) = \overline{Y_2(0)} = 0$.
- $X_5(0) = \overline{Y_2(1)} = \overline{0} = 1$ (since $X_2(k+1) = \overline{X_5(k)}$ and $f_2 = \text{NOT}$, we invert).

Reconstructed: $X(0) = [1, 0, 0, 1, 1]$. ✓

12 The Semi-Tensor Product (STP) Representation

The paper occasionally refers to the STP representation. While the algorithms in this paper *avoid* the STP (that is a key contribution), understanding what STP is helps appreciate why avoiding it matters.

12.1 The Basic Idea

The STP approach, due to Cheng [5], represents the dynamics of a BCN as a *matrix equation* by encoding Boolean values as vectors:

$$0 \longleftrightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad 1 \longleftrightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

A state $X \in S^n$ is encoded as a vector in $\{0,1\}^{2^n}$ using the Kronecker product (also called the semi-tensor product in this context). The BCN dynamics become a linear equation:

$$\mathbf{x}(k+1) = L\,\mathbf{u}(k) \ltimes \mathbf{x}(k),$$

where L is a $2^n \times 2^{n+p}$ matrix and $\ltimes$ denotes the semi-tensor product.

The Exponential Blow-Up

The state vector has 2^n entries and the transition matrix L has $2^n \times 2^{n+p}$ entries. For $n = 9$ SVs (the mammalian cell cycle), L is 512×1024—manageable. For $n = 30$, L would have $2^{30} \approx 10^9$ rows. For $n = 100$, L would have $2^{100} \approx 10^{30}$ rows—completely infeasible. This is why the STP representation "cannot be used in general to develop efficient algorithms" (quoting the paper). The graph-theoretic approach in this paper bypasses the STP entirely.

13 Summary of Notation

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

Symbol	Type / Domain	Meaning
$S = \{0, 1\}$	Set	The Boolean set
$[i, j]$	Set of integers	$\{i, i+1, \ldots, j\}$
n	Positive integer	Number of state variables (SVs)
p	Non-negative integer	Number of control inputs
m	Non-negative integer	Number of outputs
$X_i(k) \in S$	Boolean	Value of SV i at time k
$X(k) \in S^n$	Boolean vector	Full state at time k
$U(k) \in S^p$	Boolean vector	Control input at time k
$Y(k) \in S^m$	Boolean vector	Output at time k
$f_i : S^{n+p} \to S$	Boolean function	Update function for SV i
$h_j : S^n \to S$	Boolean function	Output function for output j
$G = (V, E)$	Directed graph	Dependency graph of the BCN
V	Set of vertices	$\{X_1, \ldots, X_n, U_1, \ldots, U_p\}$
E	Set of directed edges	Dependency edges
$V_s \subseteq V$	Subset	SV-vertices only: $\{X_1, \ldots, X_n\}$
$G_s = (V_s, E_s)$	Directed graph	Reduced dependency graph (SVs only)
$\mathcal{N}_{in}(v_i)$	Set	In-neighbors of v_i
$\mathcal{N}_{out}(v_i)$	Set	Out-neighbors of v_i
$e_{i \to j}$	Directed edge	Arc from v_i to v_j
w_{ij}	Walk	Walk from v_i to v_j in digraph
P_1	Property	Every hidden SV has a sole-dependency relay
P_2	Property	Every hidden cycle has an exit
O^i	Ordered set	The i-th disjoint observed path
N_i	Positive integer	Length of observed path O^i
$v_p \mapsto v_q$	Notation	v_p is the only in-neighbor of v_q
L_1	List	SVs violating P_1 (need to be made observable)
L_2	List	Non-observable SVs satisfying P_1
L_C	List	Cycles of L_2 nodes violating P_2
$\mathcal{I}$	Subset of $[1, n]$	Indices of SVs to make directly measurable
$\ltimes$	Operation	Semi-tensor product (STP)
L	Matrix ($2^n \times 2^{n+p}$)	STP transition matrix

14 Key Takeaways

1. **The problem**: Given a BCN, determine which SVs to observe and design an observer to reconstruct the full state from partial measurements. Both sub-problems are NP-hard in general.
2. **The approach**: Use graph-theoretic properties (P_1 and P_2) of the dependency graph to provide a *sufficient condition* for observability. This condition can be checked in polynomial time.
3. **Algorithm 1**: Decomposes the dependency graph into disjoint observed paths in $O(n)$ time (after graph construction). Each path acts as a shift register that reveals the initial values of its SVs over time.

4. **The Disjoint-Path Observer**: Uses the path decomposition to reconstruct the initial state $X(0)$ from the output sequence. Total complexity is $O(n^2 + pn)$—polynomial in the number of SVs.

5. **Algorithm 2**: Selects which SVs to make directly observable to guarantee the sufficient condition. Runs in $O(|V| + |E|)$ time. Sub-optimal in general, but optimal for conjunctive BCNs.

6. **The trade-off**: The sufficient condition may require more outputs than the true minimum, but the polynomial complexity makes the approach feasible for large-scale networks where exponential-time methods (STP-based) are completely impractical.

7. **Biological relevance**: Demonstrated on a mammalian cell cycle BCN with 9 SVs, where Algorithm 2 finds the optimal solution (8 outputs). The approach scales to networks with hundreds or thousands of SVs, which is relevant for large gene regulatory networks.