Technical Companion to

Minimal Controllability of Conjunctive Boolean Networks is NP-Complete

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

Companion to the paper by Eyal Weiss, Michael Margaliot, and Guy Even
Automatica, 2018

Abstract

This document explains every equation, theorem, and proof step in the paper "Minimal Controllability of Conjunctive Boolean Networks is NP-Complete" in detail, assuming only undergraduate-level knowledge of discrete mathematics, basic graph theory, and introductory algorithms. No prior knowledge of Boolean networks, control theory, or NP-completeness proofs 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 result makes sense, and (4) concrete numerical examples where helpful.

Contents

1 What This Paper Is About (The Big Picture)

Imagine a network of genes in a cell. Each gene is either "on" (expressed) or "off" (not expressed), and each gene's next state depends on the current states of other genes. A natural question arises: can we add external controls to *some* of these genes so that we can steer the entire network from any starting configuration to any desired configuration?

The key questions of the paper are:

1. **How can we tell if a given controlled network is controllable?** The paper gives a simple graph-theoretic test that runs in $O(n^2)$ time.
2. **What is the minimum number of genes we need to control?** The paper proves that finding this minimum number is **NP-hard**—meaning there is (likely) no efficient algorithm for it.

Key Idea

The paper reveals a surprising contrast: *checking* whether a specific set of controls makes the network controllable is easy (polynomial time), but *finding* the smallest such set is hard (NP-complete). This is analogous to how checking whether a proposed solution to a Sudoku puzzle is valid is easy, but finding the solution itself is hard.

2 Background: Boolean Networks

2.1 What Is a Boolean Network?

A **Boolean network (BN)** is a discrete-time dynamical system where every variable (called a **state-variable**) takes values in $\{0, 1\}$.

Definition 2.1 (Boolean Network)**.** A Boolean network with n state-variables is a system:

$$X_i(k+1) = f_i\big(X_1(k), X_2(k), \ldots, X_n(k)\big), \quad i = 1, \ldots, n,$$

where each $f_i : \{0,1\}^n \to \{0,1\}$ is a Boolean function, and $k = 0, 1, 2, \ldots$ is the discrete time step.

Symbol-by-symbol breakdown:

- $X_i(k) \in \{0,1\}$: the value of the i-th state-variable at time step k. Think of it as: is gene i on (1) or off (0) at time k?
- $X_i(k+1)$: the value of X_i at the *next* time step.
- f_i: the **update function** for variable i. It takes the current values of *all* n variables and produces the next value of variable i.
- k: the discrete time index. The system evolves step by step: $k = 0$ (initial state), $k = 1, 2, \ldots$

The **state** of the entire network at time k is the vector:

$$X(k) = \begin{bmatrix} X_1(k) \\ X_2(k) \\ \vdots \\ X_n(k) \end{bmatrix} \in \{0,1\}^n$$

Since each X_i takes values in $\{0,1\}$, there are exactly 2^n possible states.

A Simple 2-Variable Boolean Network

Consider $n = 2$ with update functions:

$$X_1(k+1) = X_2(k),$$
$$X_2(k+1) = X_1(k) \text{ AND } X_2(k).$$

Starting from $X(0) = [1, 0]'$:

k	$X_1(k)$	$X_2(k)$
0	1	0
1	$X_2(0) = 0$	$X_1(0)$ AND $X_2(0) = 1$ AND $0 = 0$
2	$X_2(1) = 0$	$X_1(1)$ AND $X_2(1) = 0$ AND $0 = 0$

The system reaches the state $[0, 0]'$ at time $k = 1$ and stays there forever. This state is called a **fixed point** or **steady state**.

2.2 What Is Controllability?

Definition 2.2 (Controllability)**.** A Boolean control network (BCN) with state-vector $X(k) = [X_1(k) \ \dots \ X_n(k)]'$ is **controllable** if for *any* pair of states $a, b \in \{0, 1\}^n$, there exists an integer $N \geq 0$ and a control sequence $u(0), \dots, u(N-1)$ steering the state from $X(0) = a$ to $X(N) = b$.

In plain English: No matter where the system starts, and no matter where you want it to end up, you can find a sequence of control inputs that drives the system from the starting state to the desired state.

Key Idea

Controllability is a very strong requirement. It demands that *every* pair of states is reachable, not just some specific pair. For a system with n variables, this means that for each of the $2^n \times 2^n$ possible (start, end) pairs, a valid control sequence must exist.

Controllability vs. Reachability

Controllability is not the same as reachability. Reachability asks: "Can I get from state a to state b?" Controllability asks: "Can I get from *any* state to *any other* state?" A system can be reachable between some pairs of states without being controllable.

2.3 What Is a Conjunctive Boolean Network (CBN)?

The paper focuses on a special class of BNs called **conjunctive Boolean networks (CBNs)**, where every update function uses only the AND operator.

Definition 2.3 (Conjunctive Boolean Network)**.** A CBN is a BN where each update function has the form:

$$X_i(k+1) = \prod_{j=1}^{n} \big(X_j(k)\big)^{\epsilon_{ji}}, \quad i = 1, \dots, n, \tag{1}$$

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: if $\epsilon_{ji} = 1$ then X_j participates; if $\epsilon_{ji} = 0$ then X_j does not (since $X_j^0 = 1$ for any X_j, it drops out of the product).

Symbol-by-symbol breakdown:

- $\prod_{j=1}^{n}$: product over all n variables. Since each factor is 0 or 1, a product of 0/1 values equals the AND of those values.
- $(X_j(k))^{\epsilon_{ji}}$: if $\epsilon_{ji} = 1$, this equals $X_j(k)$; if $\epsilon_{ji} = 0$, this equals $X_j(k)^0 = 1$ (regardless of $X_j(k)$).
- The product of all factors where $\epsilon_{ji} = 1$ gives AND of those variables, because $1 \cdot 1 \cdots 1 = 1$ only when all are 1, and any 0 makes the product 0.

Why AND Networks Are Special

The AND function has a crucial property: 0 is a **canalyzing value**. This means that if *any single* input is 0, the output is 0 regardless of all other inputs:

$$\text{AND}(0, x_1, x_2, \ldots, x_k) = 0 \quad \text{for any } x_1, \ldots, x_k \in \{0, 1\}.$$

This makes zero a "dominant" value that propagates through the network. A single zero can "infect" downstream variables, turning them to zero as well.

A 3-Variable CBN

Consider the CBN with $n = 3$:

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

Here:

- X_1's next value depends only on X_2 (so $\epsilon_{21} = 1$, all others 0).
- X_2's next value is X_1 AND X_2 (so $\epsilon_{12} = 1, \epsilon_{22} = 1$).
- X_3's next value is X_2 AND X_3 (so $\epsilon_{23} = 1, \epsilon_{33} = 1$).

Let us trace the evolution from $X(0) = [1, 1, 1]'$:

k	X_1	X_2	X_3
0	1	1	1
1	$X_2(0) = 1$	$1 \cdot 1 = 1$	$1 \cdot 1 = 1$
2	1	1	1

The all-ones state $[1, 1, 1]'$ is a fixed point. Now try $X(0) = [1, 0, 1]'$:

k	X_1	X_2	X_3
0	1	0	1
1	0	$1 \cdot 0 = 0$	$0 \cdot 1 = 0$
2	0	0	0

The zero in X_2 propagates and drives the entire system to $[0, 0, 0]'$.

2.4 The Dependency Graph

Every CBN can be represented by a **dependency graph**: a directed graph that shows which variables influence which.

Definition 2.4 (Dependency Graph)**.** Given a CBN as in Equation (1), the dependency graph $G = (V, E)$ is a directed graph with n vertices (one per state-variable) and a directed edge $(i \to j)$

if and only if $\epsilon_{ij} = 1$, meaning $X_i(k)$ appears in the update function of $X_j(k+1)$.

In plain English: Draw an arrow from variable X_i to variable X_j whenever X_i's current value affects X_j's next value.

Dependency Graph for the 3-Variable CBN

For the CBN above ($X_1(k+1) = X_2(k)$, $X_2(k+1) = X_1X_2$, $X_3(k+1) = X_2X_3$):
The edges are:

- $2 \to 1$ (because X_2 appears in X_1's update function),
- $1 \to 2$ and $2 \to 2$ (because X_1 and X_2 appear in X_2's update),
- $2 \to 3$ and $3 \to 3$ (because X_2 and X_3 appear in X_3's update).

Note that X_2 has a **self-loop** (edge $2 \to 2$) because its update function depends on its own value.

One-to-One Correspondence

When no update function is constant (i.e., every variable depends on at least one other variable), the CBN is uniquely determined by its dependency graph. The paper assumes this throughout: every vertex in the dependency graph has positive in-degree.

2.5 The Controllability Problem for CBNs

A **conjunctive Boolean control network (CBCN)** is formed by replacing some of the update functions in a CBN with external control inputs.

Definition 2.5 (CBCN — The Controlled System)**.** Given a CBN with n state-variables, suppose we choose a set $\mathcal{I} \subseteq \{1, \ldots, n\}$ of indices. For each $i \in \mathcal{I}$, we replace the update function $X_i(k+1) = \prod_j (X_j(k))^{\epsilon_{ji}}$ with $X_i(k+1) = U_i(k)$, where $U_i(k) \in \{0, 1\}$ is an independent control input. The resulting system is:

$$\begin{aligned} X_i(k+1) &= U_i(k), && i \in \mathcal{I}, \\ X_i(k+1) &= \prod_{j=1}^{n} (X_j(k))^{\epsilon_{ji}}, && i \notin \mathcal{I}. \end{aligned} \tag{2}$$

What this means: For each controlled variable $i \in \mathcal{I}$, we throw away its original update rule and instead let an external controller directly set its value at each time step. The uncontrolled variables continue to evolve according to the original AND-based rules.

The central problem of the paper:

Problem 1 (Minimal Controllability). Given a CBN with n state-variables, find a *minimum-cardinality* set $\mathcal{I} \subseteq \{1, \ldots, n\}$ such that the CBCN (2) is controllable.

In other words: what is the fewest number of variables we need to directly control so that we can steer the entire network to any desired state?

The Paper's Example 1: Making a CBN Controllable

Consider the CBN:

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

Without controls: Starting from $[0,0]'$, we get $X_1(1) = 0$, $X_2(1) = 0$. The system is stuck at $[0,0]'$ forever—we can never reach $[1,1]'$. So this CBN is *not controllable*.
With one control: Replace X_2's update with $U_2(k)$:

$$X_1(k+1) = X_2(k),$$
$$X_2(k+1) = U_2(k).$$

Now, to reach any desired state $s = [s_1, s_2]'$:

1. Set $U_2(0) = s_1$ and $U_2(1) = s_2$.
2. At $k = 1$: $X_2(1) = U_2(0) = s_1$ and $X_1(1)$ depends on the initial $X_2(0)$ (which we do not control, but that is fine—we only care about the final state).
3. At $k = 2$: $X_1(2) = X_2(1) = s_1$ and $X_2(2) = U_2(1) = s_2$.

So $X(2) = [s_1, s_2]' = s$. By controlling just one variable (X_2), we made the system controllable. The solution to Problem 1 is $\mathcal{I} = \{2\}$ with $|\mathcal{I}| = 1$.

3 Background: Graph Theory Concepts

The paper uses several concepts from graph theory. We review them here with definitions and examples.

3.1 Undirected Graphs, Neighbors, and Dominating Sets

Definition 3.1 (Undirected Graph)**.** An undirected graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. Each edge is an unordered pair $\{v_i, v_j\}$, often written (v_i, v_j). If $(v_i, v_j) \in E$, we say v_i and v_j are **neighbors**. The set of neighbors of v_i is denoted $\mathcal{N}(v_i)$, and the **degree** of v_i is $|\mathcal{N}(v_i)|$.

Definition 3.2 (Dominating Set)**.** A **dominating set** D of an undirected graph $G = (V, E)$ is a subset $D \subseteq V$ such that every vertex in $V \setminus D$ has at least one neighbor in D.

In plain English: A dominating set "covers" the entire graph in the sense that every vertex is either in the set or adjacent to something in the set.

Problem 2 (Dominating Set Problem). Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, does there exist a dominating set D of V with $|D| \leq k$?

This is a classic NP-complete problem. The paper uses it as the starting point for its NP-hardness reduction.

Dominating Set on a Small Graph

Consider a path graph with 4 vertices:

$$V = \{v_1, v_2, v_3, v_4\}, \quad E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}.$$

- $D = \{v_2, v_3\}$ is a dominating set of size 2: v_1 is adjacent to $v_2 \in D$, v_4 is adjacent to $v_3 \in D$, and v_2, v_3 are in D.
- $D = \{v_1\}$ is *not* a dominating set: v_3 and v_4 have no neighbor in D (the only neighbor of v_3 in D would need to be v_2 or v_4, neither of which is in D; note v_1 is not adjacent to v_3).
- The minimum dominating set has size 2 (e.g., $\{v_2, v_3\}$).

3.2 Directed Graphs (Digraphs)

Definition 3.3 (Directed Graph)**.** A directed graph (digraph) $G = (V, E)$ has a set of vertices V and a set of directed edges (arcs) E. Each arc $e_{i\to j}$ (or $(v_i \to v_j)$) goes from v_i to v_j. We call v_i an **in-neighbor** of v_j and v_j an **out-neighbor** of v_i.

Notation:

- $\mathcal{N}_{in}(v_i)$: the set of in-neighbors of v_i (vertices with arcs pointing *to* v_i).
- $\mathcal{N}_{out}(v_i)$: the set of out-neighbors of v_i (vertices that v_i points *to*).
- In-degree: $|\mathcal{N}_{in}(v_i)|$. Out-degree: $|\mathcal{N}_{out}(v_i)|$.
- A **source** is a vertex with in-degree zero.
- A **sink** is a vertex with out-degree zero.

Definition 3.4 (Walk, Path, Cycle)**.**

- A **walk** from v_i to v_j is a sequence of vertices $v_{i_0}, v_{i_1}, \ldots, v_{i_q}$ where $v_{i_0} = v_i$, $v_{i_q} = v_j$, and each consecutive pair is connected by an arc.
- A **simple path** is a walk with all vertices distinct.
- A **cycle** is a closed walk where all vertices are distinct (except the start equals the end).

Definition 3.5 (DAG and Strongly Connected)**.**

- A **directed acyclic graph (DAG)** is a digraph with no cycles.
- A digraph is **strongly connected** if every vertex is reachable from every other vertex.

Definition 3.6 (Multi-Layer Graph)**.** An m-layer graph is a digraph $G = (V, E)$ where V is partitioned into layers $L_1, L_2, \ldots, L_m$, and every arc goes from a vertex in layer L_k to a vertex in layer L_{k+1} for some k.

Self-Loops in Dependency Graphs

A self-loop $(v_i \to v_i)$ means that X_i's update function depends on its own current value. In the context of multi-layer graphs, a self-loop is an arc from a vertex to itself. Self-loops play an important role in the NP-hardness reduction.

4 Main Results Overview

The paper proves three main results:

1. **Theorem 6:** A CBCN is controllable if and only if its dependency graph is a DAG and satisfies Property P (defined below).
2. **Algorithm 2:** An $O(n^2)$-time algorithm to test whether a given CBCN is controllable, based on Theorem 6.
3. **Theorem 2:** The minimal controllability problem (Problem 1) is NP-hard, via a reduction from the Dominating Set problem.

The Surprising Contrast

Checking controllability of a given CBCN is easy: $O(n^2)$ time.
Finding the minimum set of controls to add is hard: NP-complete.

This is like saying: "I can quickly verify whether a given lock combination works, but finding the combination by trying all possibilities takes exponentially long."

We now walk through each result step by step, starting with the NP-hardness proof, then the characterization theorem, and finally the algorithm.

5 The NP-Hardness Proof (Theorem 2)

The paper's Theorem 2 states:

Theorem 5.1 (Theorem 2 of the paper)**.** *The decision version of Problem 1 is NP-hard.*

The decision version asks: given a CBN and a positive integer k, does there exist a set $\mathcal{I}$ with $|\mathcal{I}| \leq k$ such that the CBCN is controllable?

The proof works by **reduction from the Dominating Set problem**. We now walk through this reduction step by step.

5.1 What Is a Reduction?

Reductions 101

To prove that Problem A is NP-hard, we show that a *known* NP-hard problem (Problem B) can be transformed into Problem A in polynomial time.
The logic is:

1. If Problem A were "easy" (solvable in polynomial time),
2. then we could solve Problem B in polynomial time too (by transforming any instance of B into an instance of A and solving it),
3. but Problem B is known to be NP-hard (no polynomial-time algorithm exists unless P = NP),
4. therefore Problem A cannot be easy either—it must also be NP-hard.

Here, Problem B = Dominating Set (known NP-complete) and Problem A = Minimal Controllability of CBNs. The reduction transforms any dominating set instance into a minimal controllability instance.

5.2 Step 1: The Construction (From Graph to CBCN)

Given an undirected graph $G = (V, E)$ with $V = \{v_1, \ldots, v_s\}$ (where $s = |V|$), we construct a directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$ that will serve as the dependency graph of a CBCN.

The construction has three parts:

Part A: Create nodes.

- **Layer L_2:** For each *edge* $e_{uv} \in E$ of the original graph G, create a node in $\tilde{G}$. Also label this node e_{uv}. So $L_2 := E$, and $|L_2| = |E|$.
- **Layer L_3:** For each *vertex* $v_i \in V$ of the original graph G, create a node in $\tilde{G}$. So $L_3 := V$, and $|L_3| = |V|$.
- Total: $\tilde{V} := L_2 \cup L_3$, with $|\tilde{V}| = |E| + |V|$ nodes.

Part B: Create directed edges. $\tilde{E}$ includes two types of directed edges:

1. **Self-loops on L_2:** Every node $e_{uv} \in L_2$ gets a self-loop $(e_{uv} \to e_{uv})$.
2. **Edges from L_2 to L_3:** For each edge $e = \{u, v\} \in E$ of the original graph, create two arcs in $\tilde{G}$:
   - $(e_{uv} \to v)$: from the edge-node to vertex v.
   - $(e_{uv} \to u)$: from the edge-node to vertex u.

So $|\tilde{E}| = |E| + 2|E| = 3|E|$.

Part C: Interpret as a CBCN. The directed graph $\tilde{G}$ is the dependency graph of a CBN. Each node's update function is the AND of all its in-neighbors' values.

The Paper's Example 2: The Full Construction

Consider the undirected graph G:

$$V = \{v_1, v_2, v_3, v_4\}, \quad E = \{e_{13}, e_{23}, e_{34}\},$$

where $e_{13} = \{v_1, v_3\}$, $e_{23} = \{v_2, v_3\}$, $e_{34} = \{v_3, v_4\}$.

Step 1 (Nodes):

- $L_2 = \{e_{13}, e_{23}, e_{34}\}$ (3 edge-nodes).
- $L_3 = \{v_1, v_2, v_3, v_4\}$ (4 vertex-nodes).
- Total: 7 nodes.

Step 2 (Edges):

- Self-loops: $e_{13} \to e_{13}$, $e_{23} \to e_{23}$, $e_{34} \to e_{34}$.
- From e_{13}: arcs $e_{13} \to v_1$ and $e_{13} \to v_3$.
- From e_{23}: arcs $e_{23} \to v_2$ and $e_{23} \to v_3$.
- From e_{34}: arcs $e_{34} \to v_3$ and $e_{34} \to v_4$.
- Total: $3 + 6 = 9 = 3 \times |E| = 3 \times 3$ arcs.

This is a **2-layer graph**: all arcs go from L_2 to L_3 (plus self-loops within L_2).

The resulting CBN has 7 state-variables:

- For e_{13}: $X_{e_{13}}(k+1) = X_{e_{13}}(k)$ (only in-neighbor is itself).
- For e_{23}: $X_{e_{23}}(k+1) = X_{e_{23}}(k)$ (self-loop only).
- For e_{34}: $X_{e_{34}}(k+1) = X_{e_{34}}(k)$ (self-loop only).
- For v_1: $X_{v_1}(k+1) = X_{e_{13}}(k)$ (only in-neighbor is e_{13}).
- For v_2: $X_{v_2}(k+1) = X_{e_{23}}(k)$ (only in-neighbor is e_{23}).
- For v_3: $X_{v_3}(k+1) = X_{e_{13}}(k) \cdot X_{e_{23}}(k) \cdot X_{e_{34}}(k)$ (AND of all three edge-nodes).
- For v_4: $X_{v_4}(k+1) = X_{e_{34}}(k)$ (only in-neighbor is e_{34}).

5.3 Step 2: Why Controls Are Needed on All of L_2

Key Idea

Every node in L_2 has a self-loop, which means it remembers its own value forever. Without a control input, such a node can never change its value: if it starts at 0, it stays at 0 forever (since $X(k+1) = X(k)$). Therefore, *every* node in L_2 must receive a control input.

This means the controls on L_2 are "free"—they are mandatory for controllability, and their cost is the same for every solution. The interesting question is: which nodes in L_3 also need controls?

When we add a control to each node in L_2, we get a 3-layer CBCN:

- **Layer 1** (L_1): the control input nodes $U_{e_{uv}}$ for each $e_{uv} \in L_2$. Each control has a single arc to its corresponding node in L_2.
- **Layer 2** (L_2): the edge-nodes (now directly controlled).
- **Layer 3** (L_3): the vertex-nodes (some may also need controls).

The solution to the *minimal* controllability problem may also add controls to some nodes $Y \subseteq L_3$. Since $L_3 = V$ (the vertices of the original graph), the set Y is also a set of vertices in G.

5.4 Step 3: Connecting Controllability to Domination (Lemma 1)

The crucial link between the controllability problem and the dominating set problem is established through Lemma 1:

Lemma 5.2 (Lemma 1 of the paper)**.** *Consider a CBCN with a dependency graph that is a 3-layer graph satisfying: every vertex in layer 1 is a control input to a vertex in layer 2, and every vertex in layer 2 has its own control input (in layer 1). This CBCN is controllable if and only if every vertex in layer 3 has an in-neighbor (in layer 2) with out-degree equal to one.*

What it says in plain English: For this specific 3-layer structure, the CBCN is controllable precisely when every vertex in the bottom layer (L_3) has at least one "private pipeline" from L_2—an in-neighbor that points *only* to it and nowhere else in L_3.

Why does this condition matter?

The Out-Degree-One Condition

Recall that the update functions use AND, and 0 is the canalyzing value. To set a vertex $v \in L_3$ to 0, we need at least one of its in-neighbors in L_2 to be 0. But if that in-neighbor also points to other vertices in L_3, setting it to 0 would force those other vertices to 0 as well, even if we want them to be 1.
The out-degree-one condition ensures that for every $v \in L_3$, there exists an in-neighbor in L_2 that points *only* to v. This gives us a "private channel" to set v to 0 independently, without affecting other vertices.

The Out-Degree Condition in Action

Continuing the example with $G = (V, E)$ where $E = \{e_{13}, e_{23}, e_{34}\}$:
After adding controls to all of L_2, consider the out-degrees of the L_2 nodes *into* L_3 (ignoring self-loops):

- e_{13}: points to v_1 and v_3 $\Rightarrow$ out-degree 2 into L_3.
- e_{23}: points to v_2 and v_3 $\Rightarrow$ out-degree 2 into L_3.
- e_{34}: points to v_3 and v_4 $\Rightarrow$ out-degree 2 into L_3.

No node in L_2 has out-degree 1 into L_3! So Lemma 1 tells us that this CBCN is *not* controllable (with controls only on L_2).
Now suppose we add a control to $v_3 \in L_3$. In the dependency graph, this means we remove all arcs pointing to v_3 (from e_{13}, e_{23}, e_{34}) and replace them with a single arc from a new control node U_{v_3} to v_3.
After this modification:

- e_{13}: now points only to v_1 $\Rightarrow$ out-degree 1 into L_3.
- e_{23}: now points only to v_2 $\Rightarrow$ out-degree 1 into L_3.
- e_{34}: now points only to v_4 $\Rightarrow$ out-degree 1 into L_3.

Now every vertex in L_3 has an in-neighbor with out-degree 1: v_1 has e_{13}, v_2 has e_{23}, v_3 has U_{v_3} (the control), and v_4 has e_{34}. By Lemma 1, the CBCN is now controllable.
The set $Y = \{v_3\}$ corresponds to the dominating set $D = \{v_3\}$ in G, because v_3 is adjacent to all other vertices.

5.5 Step 4: Proof of Lemma 1 (Detailed Walkthrough)

We now explain both directions of the proof.

5.5.1 Forward Direction: Controllable ⇒ Every L_3 Vertex Has an Out-Degree-1 In-Neighbor

The proof proceeds by contradiction.

Assume: The CBCN is controllable, but there exists a vertex v_i in L_3 such that *none* of its in-neighbors in L_2 has out-degree 1.

What this means: Every in-neighbor of v_i in L_2 also points to at least one *other* vertex in L_3.

Goal: Show this leads to a contradiction.

The argument:

1. If v_i has no in-neighbors at all, then $X_i(k)$ is constant—a contradiction with controllability.
2. So v_i has at least one in-neighbor, and each in-neighbor has out-degree > 1, meaning it also points to at least one other vertex in L_3.
3. Consider the target state where $X_i(T) = 0$ and $X_j(T) = 1$ for all $j \neq i$.
4. Since the update function of v_i is an AND of its in-neighbors' values: $X_i(T) = 0$ requires at least one in-neighbor to be 0 at time $T - 1$.
5. But that in-neighbor (call it e) also points to some other vertex v_q in L_3 with $q \neq i$, and $X_q(T)$ is the AND of *its* in-neighbors. Since e is 0 at time $T - 1$, $X_q(T) = 0$ as well.
6. This contradicts the requirement $X_q(T) = 1$!

Why the Contradiction Works

The AND function with its canalyzing value of 0 means that setting any input to 0 forces the output to 0. If the only way to make v_i zero requires making an in-neighbor zero, and that in-neighbor is shared with v_q, then v_q is also forced to zero. You cannot independently set $v_i = 0$ and $v_q = 1$.

5.5.2 Backward Direction: Every L_3 Vertex Has an Out-Degree-1 In-Neighbor ⇒ Controllable

This direction constructs an explicit control sequence to steer the CBCN from any initial state to any desired final state.

Setup: Let the nodes in L_3 be $w_1, \ldots, w_q$ and suppose node w_i has an in-neighbor $v_i \in L_2$ with out-degree 1 (pointing only to w_i). Let the remaining nodes in L_2 be $v_{q+1}, \ldots, v_p$ with controls $u_1, \ldots, u_p$.

Desired final state: $v = [v_1 \ldots v_p]'$ for L_2 nodes and $w = [w_1 \ldots w_q]'$ for L_3 nodes.

Fix arbitrary $a \in \{0,1\}^p$ (desired for L_2) and $b \in \{0,1\}^q$ (desired for L_3).

The control sequence:

1. **Times 0 and 1:** Set all controls to 1. After 2 steps, all state-variables in L_2 and L_3 equal 1 (because AND of all-ones is one).
2. **Time 2:** Set $u_i(2) = b_i$ for $i = 1, \ldots, q$ and $u_i(2) = 1$ for $i > q$. This propagates desired values through the out-degree-1 paths.
3. **Time 3:** Set $u_i(3) = a_i$ for all i. Now $v_i(3) = b_i$ for $i \leq q$, and $w_i(4) = b_i$ (because w_i's private in-neighbor delivers b_i, and $\text{AND}(1, b_i) = b_i$).

4. At $k = 4$: $w_i(4) = b_i$ for all i and $v_i(4) = a_i$ for all i.

The key trick is that the out-degree-1 in-neighbors act as "private channels" that can independently deliver any desired value to each L_3 node.

Constructing a Control Sequence

For the modified CBCN with $Y = \{v_3\}$ controlled:

- L_2: e_{13} (private to v_1), e_{23} (private to v_2), e_{34} (private to v_4).
- L_3: v_1, v_2, v_3 (with v_3 controlled), v_4.

Goal: steer to $[v_1, v_2, v_3, v_4] = [0, 1, 1, 0]$.

k	Controls	e_{13}	e_{23}	e_{34}
0–1	all $= 1$	1	1	1
2	$u_{e_{13}} = 0, u_{e_{23}} = 1, u_{e_{34}} = 0$	0	1	0

At $k = 3$: $v_1 = e_{13}(2) = 0$, $v_2 = e_{23}(2) = 1$, $U_{v_3}(2) = 1$ so $v_3 = 1$, $v_4 = e_{34}(2) = 0$. Result: $[0, 1, 1, 0]$ as desired.

5.6 Step 5: Connecting Y to a Dominating Set (Lemma 3)

Lemma 5.3 (Lemma 3 of the paper)**.** *The set Y (the vertices in L_3 that receive controls in the optimal solution) is a minimum dominating set of the original graph G.*

Why is this true?

Recall the construction: for each edge $\{u, v\} \in E$ in the original graph, there is an edge-node $e_{uv} \in L_2$ with arcs to both u and v in L_3.

When we add a control to a vertex $w \in L_3$ (i.e., $w \in Y$), we remove all arcs pointing to w from L_2 nodes. This may reduce the out-degree of some L_2 nodes from 2 to 1, creating the "private channels" needed for Lemma 1.

The key observation: Consider a vertex $v \in V \setminus Y$ (a vertex in the original graph that does *not* get controlled). For the CBCN to be controllable (by Lemma 1), v must have an in-neighbor $e \in L_2$ with out-degree 1.

The edge-node $e = e_{uv}$ originally points to both u and v. For e to have out-degree 1 (pointing only to v), the arc from e to u must have been removed—which happens only if $u \in Y$ (i.e., u received a control).

But u is a neighbor of v in the original graph G (since the edge $\{u, v\}$ exists). So every vertex $v \notin Y$ has a neighbor in Y, which is exactly the definition of Y being a dominating set!

The Bijection Between Controls and Domination

Controllability problem	$\longleftrightarrow$	**Dominating set problem**
Vertex $v \in L_3$ gets a control	$\longleftrightarrow$	Vertex $v \in D$
Edge-node e_{uv} has out-degree 1	$\longleftrightarrow$	Edge $\{u, v\}$ "covers" v
Every L_3 node has a private in-neighbor from L_2	$\longleftrightarrow$	Every vertex is dominated

Adding a control to v "frees up" the edge-nodes incident to v, reducing their out-degree and creating private channels for v's neighbors. This is exactly what a dominating set vertex does: it "covers" all its neighbors.

5.7 Step 6: Completing the Proof of Theorem 2

The proof finishes by observing:

1. The construction (from G to $\tilde{G}$) runs in polynomial time: we create $|E| + |V|$ nodes and $3|E|$ arcs.
2. Solving the minimal controllability problem for the CBCN yields a set $Y \subseteq L_3$ that is a minimum dominating set of G (by Lemma 3).
3. If $|Y| \leq k$, the answer to the dominating set instance is "yes"; if $|Y| > k$, the answer is "no."
4. Since the Dominating Set problem is NP-hard, the minimal controllability problem must also be NP-hard.

NP-Hard vs. NP-Complete

The paper proves NP-*hardness*, meaning the problem is at least as hard as any problem in NP. To show NP-*completeness*, one also needs to show that the problem is *in* NP (i.e., a proposed solution can be verified in polynomial time). Indeed, given a proposed set $\mathcal{I}$, we can verify controllability in $O(n^2)$ time using the algorithm from the paper, so the problem is also in NP, making it NP-complete.
The paper's title says "NP-Complete" even though Theorem 2 technically proves NP-hardness; the "in NP" direction follows from Algorithm 2.

6 Necessary and Sufficient Conditions for Controllability

Before presenting the $O(n^2)$ algorithm, the paper develops the graph-theoretic characterization of controllable CBCNs. This section explains Propositions 4, 5 and Theorem 6.

6.1 Terminology for CBCN Dependency Graphs

The paper introduces specific terminology for nodes in the dependency graph of a CBCN:

Term	Definition	Properties
Simple node	Represents a state-variable (SV)	Has in-neighbors
Generator	Represents a control input	Always a source; has exactly one out-neighbor (a simple node)
Directly controlled node	A simple node with an added control input	Also has a generator as an in-neighbor
Channel	A simple node with out-degree 1, no self-loop	Only out-neighbor is another simple node

Identifying Node Types

Consider the CBCN from Equation (4) of the paper:

$$X_1(k+1) = U_1(k),$$
$$X_2(k+1) = U_2(k),$$
$$X_3(k+1) = X_1(k) \cdot X_2(k).$$

- U_1, U_2 are **generators** (control inputs; sources).
- X_1, X_2 are **directly controlled nodes** (simple nodes with generator in-neighbors).
- X_3 is a **simple node** (but not directly controlled).
- X_1 is also a **channel**: its only out-neighbor is X_3, and it has no self-loop.
- Similarly, X_2 is a channel.

6.2 Proposition 4: The Dependency Graph Must Be a DAG

Proposition 6.1 (Proposition 4 of the paper)**.** *The dependency graph of a controllable CBCN is acyclic (a DAG).*

What it says in plain English: If the dependency graph of a CBCN contains a cycle, the system cannot be controllable.

Why is this true? (Proof sketch)

1. Every vertex in a cycle is a simple node (not a generator, since generators are sources with no incoming arcs, and every vertex in a cycle has at least one incoming arc from within the cycle).
2. Moreover, no simple node in a cycle can be directly controlled, because a directly controlled node's only in-neighbor is its generator (a source), which cannot be part of a cycle.
3. Now consider starting the system with all variables in the cycle set to 0.
4. Since the update function of each node in the cycle is an AND that includes at least one other node in the cycle, and all those nodes are 0, the outputs remain 0.
5. There is no way to "break into" the cycle with a control input—the cycle variables stay at 0 forever.
6. Therefore we cannot steer to the state where all cycle variables are 1: the system is not controllable.

A Cycle Breaks Controllability

Consider the CBN with a 2-cycle:

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

Starting from $[0,0]'$: $X_1(1) = X_2(0) = 0$, $X_2(1) = X_1(0) = 0$. The system stays at $[0,0]'$ forever. There is no external control and no way to inject a 1 into the cycle.
Even if we add a control somewhere else (say a third variable X_3), the cycle between X_1 and X_2 is self-contained—we cannot steer it to $[1,1]'$ without directly controlling one of its members.

6.3 Proposition 5: Property P Is Necessary

Definition 6.1 (Property P)**.** A CBCN *has Property P* if every simple node in its dependency graph has at least one in-neighbor that is either a generator or a channel.

In plain English: Every state-variable has at least one "clean input path" through which a specific value can be delivered without side effects. A generator directly injects any value. A channel passes along values from its single predecessor without fan-out.

Proposition 6.2 (Proposition 5 of the paper)**.** *A controllable CBCN has Property P.*

Why is this true? (Proof by contradiction)

Assume the CBCN is controllable but there exists a simple node v in its dependency graph that does not contain a generator nor a channel in its set of in-neighbors.

1. **What "no generator or channel" means structurally.** A generator is a source with out-degree 1. A channel is a simple node with out-degree 1 and no self-loop. Since v has no such in-neighbor, there does not exist a node w in the graph such that v is the *only* simple node in the out-neighbors of w.
2. **Consequence: v cannot reach zero independently.** Hence, the SV that corresponds to v cannot change its value to zero (the canalyzing value) without at least one other SV changing its value to zero as well. This is because every in-neighbor of v also feeds at least one other simple node, so setting any in-neighbor to zero forces another SV to zero too.
3. **Set up the contradiction.** Consider two states: a corresponding to all SVs being zero, and b corresponding to all SVs being one except for v that is zero. Since the CBCN is controllable, it is possible to steer it from a to b.
4. **Derive a self-loop.** This implies that v has a self-loop, as it holds the value zero while the other SVs change their values (from zero to one). A self-loop means v's own value appears in its update function, so if $v = 0$ then $\text{AND}(0, \ldots) = 0$, helping v remain at zero while others become one.
5. **Contradiction via Proposition 4.** But a self-loop is a cycle of length one. By Proposition 4, the dependency graph of a controllable CBCN is acyclic. This is a contradiction.

Generators and Channels: The "Clean Pipes"

Generators and channels serve as "clean pipes" through which values can be delivered to specific state-variables without unintended side effects.

- A **generator** (control input) can inject any value at any time.
- A **channel** has out-degree 1 and no self-loop, so the value it carries affects exactly one downstream variable.

Without at least one such clean pipe feeding each variable, independent control is impossible.

6.4 Theorem 6: The Complete Characterization

Theorem 6.3 (Theorem 6 of the paper)**.** *A CBCN is controllable if and only if its dependency graph is a DAG and satisfies Property P.*

The "only if" direction follows from Propositions 4 and 5. The "if" direction requires a constructive proof showing that a CBCN whose dependency graph is a DAG with Property P is indeed controllable.

6.4.1 The Concept of Controlled Paths

To prove the "if" direction, the paper introduces the notion of a **controlled path**.

Definition 6.2 (Controlled Path)**.** A controlled path is an ordered non-empty set of nodes in the dependency graph such that:

1. The first element is a generator.
2. If the set has more than one element, then for $i > 1$, the i-th node is a simple node.
3. For $i > 1$, the i-th node is the *only* element in the out-neighborhood of node $i - 1$ (i.e., node $i - 1$ has out-degree 1 and points only to node i).

In plain English: A controlled path is a chain of nodes starting from a control input (generator), passing through a sequence of channels (nodes with out-degree 1), and ending at a simple node. Values injected at the generator "flow down" the path one step at a time, like items on a conveyor belt.

Controlled Paths as Shift Registers

Consider the CBCN from Example 3 of the paper:

$$\begin{aligned} X_1(k+1) &= U_1(k), \\ X_2(k+1) &= U_2(k), \\ X_3(k+1) &= X_1(k) \cdot X_2(k). \end{aligned}$$

The controlled paths are:

- $C^1 = \{U_1 \to X_1 \to X_3\}$: generator U_1 feeds X_1 (which has out-degree 1, pointing only to X_3), and X_1 feeds X_3.
- $C^2 = \{U_2 \to X_2\}$: generator U_2 feeds X_2. (X_2 has out-degree 1, but X_3 is the endpoint of path C^1, so this path ends at X_2.)

Each controlled path acts like a **shift register**: at each time step, each node passes its value to the next node in the path. By feeding appropriate values into the generator, we can load any desired sequence into the path.

For $C^1 = \{U_1 \to X_1 \to X_3\}$ (length 3):

k	$U_1(k)$	$X_1(k+1)$	$X_3(k+2)$
0	a	a	
1	b	b	depends on $X_1(1)$ and $X_2(1)$

The value a injected at time 0 arrives at X_1 at time 1, and at X_3 at time 2 (as one of its AND inputs).

6.4.2 Decomposition Into Disjoint Controlled Paths (Proposition 7)

Proposition 6.4 (Proposition 7 of the paper)**.** *If the dependency graph G of a CBCN is a DAG and satisfies Property P, then G can be decomposed into disjoint controlled paths that cover every vertex.*

The paper provides Algorithm 1 for this decomposition. We reproduce it here in full, then explain its logic step by step.

Algorithm 1 Decompose the nodes in G into disjoint controlled paths

```
1: while there exists a simple node v ∈ G that is not included in any ordered set do
2:   cnode ← v and cset ← {v}
3:   if N_in(cnode) contains a channel then
4:     pick a channel u ∈ N_in(cnode)
5:     if u does not belong to any ordered set then
6:       insert u to the ordered set cset just before cnode
7:       cnode ← u
8:       goto 3
9:     else
10:      let H denote the ordered set that contains u
11:      merge cset into H keeping the order between any two adjacent elements
12:      goto 1
13:    end if
14:  else
15:    pick a generator u ∈ N_in(cnode)
16:    if u does not belong to any ordered set then
17:      insert it to cset just before cnode
18:      goto 1
19:    else
20:      goto 10
21:    end if
22:  end if
23: end while
```

How Algorithm 1 works, step by step:

The algorithm builds controlled paths by walking *backwards* from uncovered simple nodes toward generators:

1. **Pick an uncovered simple node** v and start a new ordered set $cset = \{v\}$ (line 2).
2. **Walk backwards through channels** (lines 3–8): if the current node's in-neighbors include a channel u not yet in any ordered set, prepend u to *cset* and continue walking backwards from u.
3. **Merge if the channel is already assigned** (lines 9–12): if the channel u already belongs to another ordered set H, merge *cset* into H (maintaining order) and restart the outer loop.
4. **Terminate at a generator** (lines 13–17): if no channel is available among the in-neighbors, Property P guarantees that a generator u exists. Prepend u to *cset* to complete the controlled path, and restart the outer loop.
5. **Handle already-assigned generators** (lines 18–19): if the generator is already in another set, merge as in the channel case.

Note that the assumption that G has Property P is used in line 14 of the algorithm: when no channel is available, a generator must exist among the in-neighbors. The proof that Algorithm 1 terminates in a finite number of steps and divides all the nodes into disjoint controlled paths is straightforward and thus omitted. The basic idea is that Property P and the fact that the graph is a DAG imply that we can always "go back" from any simple node through a chain of channels ending with a generator, thus creating a controlled path.

Corollary 6.5 (Corollary 8 of the paper)**.** *A CBCN is controllable if and only if its dependency graph can be decomposed into a set of disjoint controlled paths.*

6.4.3 Proof Sketch: DAG + Property P ⇒ Controllable

Given the decomposition into controlled paths $C^1, \ldots, C^m$:

1. **View each path as a shift register.** Values injected at the generator flow down the path, arriving at the i-th node after $i - 1$ time steps.
2. **First, pump all ones.** Set all control inputs to 1 for enough time steps to fill all paths with ones. Since the only Boolean function is AND, and AND(1, 1, ..., 1) = 1, all state-variables become 1.
3. **Then, load desired values.** Use each generator to feed the desired values into its path. Because the paths are disjoint (no node belongs to two paths), loading one path does not interfere with another. The fact that each non-path arc goes from the *last* node of one path to a node in another (and the DAG property ensures no circular dependencies) means that the values can be loaded in the correct order.
4. **After enough time steps, the system reaches the desired state.** The total time is proportional to the length of the longest path.

The Shift Register Analogy

Each controlled path is a shift register:

- The generator is the input to the register.
- Each channel is a flip-flop that passes its value to the next stage.
- Values shift down the path one step per time unit.
- After L time steps (where L is the path length), the value injected at the generator has reached the end of the path.

Because the AND function with all-1 inputs is the identity (AND$(1, x) = x$), once we have "primed" the system to all ones, we can treat each path as a transparent pipeline.

7 The Polynomial-Time Algorithm (Algorithm 2)

Algorithm 2 tests whether a given CBCN is controllable, based on Theorem 6.

7.1 The Algorithm

We reproduce the paper's Algorithm 2 verbatim:

Algorithm 2 Testing controllability of a CBCN in the form (2) with n SVs and $q \leq n$ control inputs

```
1: generate the dependency graph G = (V, E)
2: if G is not a DAG then
3:    return ("not controllable")
4: end if
5: create a list L of n bits
6: set all bits in L to 0
7: for all nodes v ∈ V do
8:    if |N_out(v)| ≠ 1 then
9:       return ("not controllable")
10:   else
11:      j ← the element in N_out(v)
12:      L(j) ← 1
13:   end if
14: end for
15: if all bits in L are 1 then
16:   return ("controllable")
17: else
18:   return ("not controllable")
19: end if
```

What each part does:

- **Line 1:** Build the dependency graph. The resulting graph has $|V| = n + q \leq 2n$ nodes (simple nodes plus generators) and $|E| \leq n^2$ arcs.
- **Lines 2–3:** Check Proposition 4's necessary condition (DAG).
- **Lines 4–12:** Check Property P. The loop visits every node $v \in V$ (both generators and simple nodes). If *any* node has out-degree $\neq 1$, the algorithm immediately returns "not controllable." Otherwise, the unique out-neighbor j of v is marked in L.
- **Lines 13–16:** If every SV (every entry of L) has been marked, then every SV has an in-neighbor with out-degree 1 (i.e., a generator or channel), so Property P holds.

7.2 Why This Works

- **DAG check (line 2):** Proposition 4 says a controllable CBCN must have an acyclic dependency graph.
- **Out-degree check (lines 7–12):** The loop rejects the CBCN if any node has out-degree $\neq 1$. A generator always has out-degree exactly 1 (it feeds a single simple node). A channel has out-degree exactly 1 by definition. A directly controlled node also has out-degree 1 in a controllable CBCN. The bit array L tracks which SVs have an in-neighbor with out-degree 1. Setting $L(j) = 1$ when we find a node with out-degree 1 pointing to j is equivalent to checking that j has a channel or generator as an in-neighbor.
- **Property P check (lines 13–16):** If all bits in L are 1, then every SV has a generator or channel in-neighbor, so Property P holds.
- By Theorem 6, DAG + Property P $\Leftrightarrow$ controllable.

7.3 Time Complexity

- Building the dependency graph: $O(n^2)$ (reading n update functions, each with up to n arguments).
- Topological sort: $O(|V| + |E|) = O(n + n^2) = O(n^2)$ since $|V| \leq 2n$ and $|E| \leq n^2$.
- Property P check: $O(n + q) = O(n)$ where $q \leq n$ is the number of control inputs.
- **Total:** $O(n^2)$.

Running Algorithm 2 on a Controllable CBCN

Consider the CBCN from Example 1 of the paper (after adding one control):

$$X_1(k+1) = X_2(k),$$
$$X_2(k+1) = U_2(k).$$

Line 1: Dependency graph has nodes $V = \{U_2, X_1, X_2\}$ and arcs $\{U_2 \to X_2,\ X_2 \to X_1\}$.
Line 2: Topological order: U_2, X_2, X_1. No cycles—it is a DAG.
Lines 4–5: $L = [0, 0]$ (one bit per SV: X_1, X_2).
Lines 7–12: Check out-degrees of all nodes:

- U_2: $|\mathcal{N}_{out}(U_2)| = 1$ (points to X_2). Set $L(2) = 1$.
- X_2: $|\mathcal{N}_{out}(X_2)| = 1$ (points to X_1). Set $L(1) = 1$.
- X_1: $|\mathcal{N}_{out}(X_1)| = 0 \neq 1$. The algorithm returns "not controllable."

Wait—but we know this CBCN is controllable (the paper says so in Example 1)! The issue is that X_1 is a sink node with out-degree 0.

Important subtlety: The paper assumes throughout that no SV in the network has a constant updating function, i.e., every vertex in the dependency graph has positive in-degree. In the original CBN $X_1(k+1) = X_2(k)$, $X_2(k+1) = X_1(k)X_2(k)$, node X_1 appears in the update function of X_2, so X_1 has out-degree 1 (it points to X_2). When we replace X_2's update with $U_2(k)$, the arc $X_1 \to X_2$ is removed, making X_1 a sink. In the paper's formulation this corresponds to X_1 not appearing in any remaining update function. The algorithm, as written, detects this and returns "not controllable."
However, the paper's Example 1 is presented informally *before* the algorithm is introduced, and the algorithm is designed for the general setting of Theorem 6 where the dependency graph structure satisfies the DAG and Property P conditions. In practice, a sink node (out-degree 0) that does not appear in any other update function simply terminates a controlled path. The algorithm as literally stated in the paper returns "not controllable" for any graph containing a node with out-degree $\neq 1$.

Let us instead run the algorithm on the CBCN from Example 3 of the paper, which the paper uses as its main illustrative example for controlled paths. Consider a modified version where X_3 feeds back into the system:

$$X_1(k+1) = U_1(k),$$
$$X_2(k+1) = U_2(k),$$
$$X_3(k+1) = X_1(k) \cdot X_2(k),$$
$$X_4(k+1) = X_3(k).$$

Line 1: Dependency graph: $V = \{U_1, U_2, X_1, X_2, X_3, X_4\}$, arcs $\{U_1 \to X_1,\ U_2 \to X_2,\ X_1 \to X_3,\ X_2 \to X_3,\ X_3 \to X_4\}$.
Line 2: Topological order exists: $U_1, U_2, X_1, X_2, X_3, X_4$. It is a DAG.
Lines 4–5: $L = [0, 0, 0, 0]$ (one bit per SV).
Lines 7–12: Check out-degrees:

- U_1: $|\mathcal{N}_{out}| = 1$ (points to X_1). $L(1) = 1$.
- U_2: $|\mathcal{N}_{out}| = 1$ (points to X_2). $L(2) = 1$.
- X_1: $|\mathcal{N}_{out}| = 1$ (points to X_3). $L(3) = 1$.
- X_2: $|\mathcal{N}_{out}| = 1$ (points to X_3). $L(3)$ already 1.
- X_3: $|\mathcal{N}_{out}| = 1$ (points to X_4). $L(4) = 1$.
- X_4: $|\mathcal{N}_{out}| = 0 \neq 1$. Return "not controllable."

Again, the algorithm rejects because of the sink node X_4. This illustrates that the algorithm, as literally stated, requires every node (including terminal nodes) to have out-degree exactly 1. In a CBCN where no SV has a constant update function, every SV appears in at least one other SV's update function, guaranteeing positive out-degree. Terminal (sink) nodes arise only when some SV does not influence any other SV's update.

Running Algorithm 2 on a Non-Controllable CBCN

Consider the CBN (no controls added):

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

Line 1: Dependency graph: $V = \{X_1, X_2\}$, arcs $\{X_2 \to X_1,\ X_1 \to X_2,\ X_2 \to X_2\}$.
Line 2: The graph contains cycles ($X_1 \to X_2 \to X_1$, and the self-loop $X_2 \to X_2$). Return "not controllable."
The algorithm correctly identifies this CBN as not controllable. No controls have been added, and the cycle prevents controllability (Proposition 4).

8 Putting It All Together: The Paper's Logical Flow

Here is a summary of the paper's logical structure:

Step	What Is Shown
1	**Proposition 4:** Controllable CBCN $\Rightarrow$ dependency graph is a DAG.
2	**Proposition 5:** Controllable CBCN $\Rightarrow$ Property P holds.
3	**Proposition 7:** DAG + Property P $\Rightarrow$ disjoint controlled path decomposition exists.
4	**Theorem 6:** CBCN is controllable $\Leftrightarrow$ DAG + Property P. (Combines Props. 4, 5, 7 and a constructive proof using shift registers.)
5	**Algorithm 2:** $O(n^2)$-time test for controllability, based on Theorem 6.
6	**Lemma 1:** For 3-layer CBCNs, controllability $\Leftrightarrow$ every L_3 vertex has an out-degree-1 in-neighbor.
7	**Lemma 3:** Optimal L_3 controls = minimum dominating set.
8	**Theorem 2:** Problem 1 is NP-hard (via reduction from Dominating Set using Lemmas 1 and 3).

The Two-Level Argument

Level 1 (Easy part): Given a specific CBCN, we can check controllability in $O(n^2)$ time by testing two graph properties: (a) acyclicity and (b) Property P. This is a clean, efficient characterization.
Level 2 (Hard part): Finding which variables to control is NP-hard. The reduction constructs a specially structured CBCN from a dominating set instance, where the optimal set of controls corresponds exactly to the minimum dominating set.

9 A Complete Worked Example: End to End

Let us work through the entire paper's argument on a concrete example.

9.1 The Original Graph

Consider the undirected graph $G = (V, E)$:

$$V = \{a, b, c\}, \quad E = \{\{a, b\}, \{b, c\}\}.$$

This is a simple path: $a - b - c$.

9.2 Step 1: Construct the CBCN

Following the construction in Section 5.2:

Nodes:

- $L_2 = \{e_{ab}, e_{bc}\}$ (one node per edge).
- $L_3 = \{a, b, c\}$ (one node per vertex).
- $|\tilde{V}| = 2 + 3 = 5$.

Arcs:

- Self-loops: $e_{ab} \to e_{ab}$, $e_{bc} \to e_{bc}$.
- From e_{ab}: $e_{ab} \to a$ and $e_{ab} \to b$.
- From e_{bc}: $e_{bc} \to b$ and $e_{bc} \to c$.
- $|\tilde{E}| = 2 + 4 = 6 = 3|E|$.

The resulting CBN (before any controls):

$$\begin{aligned} X_{e_{ab}}(k+1) &= X_{e_{ab}}(k), \\ X_{e_{bc}}(k+1) &= X_{e_{bc}}(k), \\ X_a(k+1) &= X_{e_{ab}}(k), \\ X_b(k+1) &= X_{e_{ab}}(k) \cdot X_{e_{bc}}(k), \\ X_c(k+1) &= X_{e_{bc}}(k). \end{aligned}$$

9.3 Step 2: Add Mandatory Controls on L_2

Since e_{ab} and e_{bc} have self-loops, they must be controlled:

$$\begin{aligned} X_{e_{ab}}(k+1) &= U_{e_{ab}}(k), \\ X_{e_{bc}}(k+1) &= U_{e_{bc}}(k), \\ X_a(k+1) &= X_{e_{ab}}(k), \\ X_b(k+1) &= X_{e_{ab}}(k) \cdot X_{e_{bc}}(k), \\ X_c(k+1) &= X_{e_{bc}}(k). \end{aligned}$$

9.4 Step 3: Check Controllability Without L_3 Controls

Out-degrees of L_2 nodes into L_3:

- e_{ab}: points to a and $b \Rightarrow$ out-degree 2.
- e_{bc}: points to b and $c \Rightarrow$ out-degree 2.

By Lemma 1, the CBCN is *not* controllable because no L_3 vertex has an in-neighbor with out-degree 1. (Both e_{ab} and e_{bc} have out-degree 2.)

9.5 Step 4: Find the Minimum Dominating Set

For the path graph $a - b - c$:

- $D = \{b\}$ is a dominating set: a is adjacent to b, and c is adjacent to b. Size: 1.
- $D = \{a\}$ is *not* a dominating set: c is not adjacent to a.
- The minimum dominating set is $\{b\}$ with $|D| = 1$.

9.6 Step 5: Add Control to b and Verify

Add a control U_b to vertex b in L_3:

$$\begin{aligned} X_{e_{ab}}(k+1) &= U_{e_{ab}}(k), \\ X_{e_{bc}}(k+1) &= U_{e_{bc}}(k), \\ X_a(k+1) &= X_{e_{ab}}(k), \\ X_b(k+1) &= U_b(k), \\ X_c(k+1) &= X_{e_{bc}}(k). \end{aligned}$$

Now we removed the arcs $e_{ab} \to b$ and $e_{bc} \to b$. Updated out-degrees:

- e_{ab}: points only to $a \Rightarrow$ out-degree 1.
- e_{bc}: points only to $c \Rightarrow$ out-degree 1.

Every L_3 vertex has an out-degree-1 in-neighbor: a has e_{ab}, b has U_b (generator), c has e_{bc}. By Lemma 1, the CBCN is controllable.

Total controls added: $|L_2| + |Y| = 2 + 1 = 3$. The minimum number of controls for L_3 is $|Y| = 1 = |D|$.

9.7 Step 6: Construct a Control Sequence

Goal: steer to $[X_a, X_b, X_c] = [1, 0, 1]$ from any initial state.

k	$U_{e_{ab}}$	$U_{e_{bc}}$	U_b	e_{ab}	e_{bc}	a	c
0	1	1	–	?	?	?	?
1	1	1	0	1	1	?	?
2	–	–	–	1	1	1	1

At $k = 2$: $X_a = e_{ab}(1) = 1$, $X_b = U_b(1) = 0$, $X_c = e_{bc}(1) = 1$. Result: $[1, 0, 1]$ as desired.

10 Discussion and Broader Context

10.1 Why CBNs? Biological Motivation

Conjunctive Boolean networks model **synergistic regulation** in gene regulatory networks. When a gene requires *all* of its transcription factors to be active in order to be expressed,

this is naturally captured by the AND function.

For example, if gene G_3 requires both transcription factor G_1 and transcription factor G_2 to be present for expression, the update rule is:

$$G_3(k+1) = G_1(k) \text{ AND } G_2(k).$$

The controllability question then becomes: which genes do we need to be able to externally manipulate (e.g., via drugs or gene therapy) so that we can drive the gene regulatory network to any desired expression pattern?

10.2 Disjunctive Boolean Networks (DBNs)

The paper notes that its results also apply to **disjunctive Boolean networks (DBNs)**, where every update function uses only OR. This is because De Morgan's laws allow us to convert any DBN into a CBN:

$$\text{OR}(X_1, X_2) = \text{NOT}(\text{AND}(\text{NOT}(X_1), \text{NOT}(X_2))).$$

By complementing all variables ($Y_i = 1 - X_i$), a DBN becomes a CBN. So the NP-hardness result applies equally to DBNs.

10.3 Comparison with LTI Systems

In continuous-time linear time-invariant (LTI) systems, the minimal controllability problem is also NP-hard (Olshevsky, 2014). The paper extends this hardness result to the discrete, Boolean setting with AND-only dynamics.

Interestingly, both results rely on reductions from similar graph problems: Olshevsky's proof uses the minimum hitting set problem, while this paper uses the minimum dominating set problem. Both are NP-complete.

10.4 Connections to Path Cover Problems

The concept of disjoint controlled paths is related to the **minimum path cover problem (MPCP)**: finding the minimum number of vertex-disjoint paths that cover all vertices in a DAG.

However, the controlled path decomposition has a stricter structure than a generic path cover:

- Each path must start at a generator (source).
- The intermediate nodes must be channels (out-degree 1).
- The AND function imposes constraints on how paths interact.

While the MPCP for DAGs can be solved in polynomial time, the *minimum controlled path cover* (i.e., finding the fewest generators to add) is NP-hard—this is precisely what the paper proves.

11 Summary of Notation

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

Symbol	Type / Domain	Meaning
n	Positive integer	Number of state-variables in the CBN
$X_i(k)$	$\in \{0,1\}$	Value of state-variable i at time k
$X(k)$	$\in \{0,1\}^n$	State vector at time k
$U_i(k)$	$\in \{0,1\}$	Control input for variable i at time k
ϵ_{ji}	$\in \{0,1\}$	Dependency indicator: 1 if X_j appears in X_i's update function
$\mathcal{I}$	$\subseteq \{1,\ldots,n\}$	Set of indices of controlled variables
$G = (V, E)$	Undirected graph	Input graph for Dominating Set
$\tilde{G} = (\tilde{V}, \tilde{E})$	Directed graph	Constructed dependency graph
L_1, L_2, L_3	Vertex sets	Layers in the 3-layer graph
$\mathcal{N}_{in}(v)$	Set of vertices	In-neighbors of vertex v
$\mathcal{N}_{out}(v)$	Set of vertices	Out-neighbors of vertex v
DAG	–	Directed Acyclic Graph
Property P	Graph property	Every simple node has a generator or channel in-neighbor
Generator	Node type	Source node representing a control input
Channel	Node type	Simple node with out-degree 1, no self-loop
Simple node	Node type	Node representing a state-variable
Directly controlled	Node type	Simple node with a generator in-neighbor
Controlled path	Ordered node set	Generator $\rightarrow$ channels $\rightarrow$ endpoint
D	$\subseteq V$	Dominating set
k	Non-negative integer	Discrete time index
$s = \lvert V \rvert$	Positive integer	Number of vertices in graph G
Y	$\subseteq L_3$	Vertices receiving additional controls

12 Key Takeaways

1. **CBNs use only AND:** Every update function is a product (AND) of selected state-variables. This makes 0 a "dominant" (canalyzing) value.
2. **Controllability = DAG + Property P:** A CBCN is controllable if and only if its dependency graph is a DAG and every state-variable has a "clean pipeline" (generator or channel) as an in-neighbor.
3. **Controlled paths = shift registers:** The key structural insight is that a controllable CBCN can be decomposed into independent shift registers that can be loaded with arbitrary values.
4. **Checking is easy, optimizing is hard:** Testing a given CBCN for controllability takes $O(n^2)$ time. But finding the minimum number of controls to add is NP-complete.
5. **The reduction goes through dominating sets:** The NP-hardness proof transforms an arbitrary dominating set instance into a minimal controllability instance via a clever 3-layer graph construction where edges become nodes and vertices become nodes in a different layer.
6. **Results extend to DBNs:** By De Morgan's laws, all results apply equally to disjunctive (OR-only) Boolean networks.