A Systematic Study of Single-Anchor Logical Gadgets

Fikret H. Güngör

November 2025

Abstract

We present a systematic study of logical gadgets for 3-coloring under a single anchor constraint, where only one color representing logical falsehood is fixed to a vertex. We introduce a framework of what we call ladgets (logical gadgets), graph gadgets that implement Boolean functions. Then, we define a set of core gadgets, called primitives, which help identify and analyze the logical behavior of ladgets. Next, we examine the structure of several standard ladgets and present several structural constraints for ladgets. Through an exhaustive search of all non-isomorphic connected graphs up to 10 vertices, we verify all minimal constructions for standard ladgets. Notably, we identify exactly two non-isomorphic minimal XNOR ladgets in approximately 29 billion gadget configurations, highlighting the rarity of gadgets capable of expressing logical behavior. We also present an embedding technique that embeds ladgets with less than 3 inputs from 3-coloring into k-coloring. Our work shows how the single anchor constraint creates a fundamentally different framework from the two anchor gadgets used in SAT reductions.

1 Introduction

The 3-coloring problem is one of the most fundamental NP-complete problems, asking whether a graph can be colored with 3 colors. Reductions of this problem involve developing *logical gadgets*, small graphs that simulate Boolean operations within some coloring constraints.

Traditional approaches for developing logical gadgets generally use two anchors, two vertices fixed to colors representing truth and falsehood in the logical sense. While this approach results in smaller and easier gadgets to work with because of the symmetry, it involves setting too many color constraints therefore it feels unnatural from a graph-theoretic perspective. This raises this natural question: *What happens when we break the symmetry?*

In this paper, we investigate logical gadgets under a *single anchor constraint*, fixing a single color representing falsehood to a distinguished *anchor vertex*, while truth values represented by two remaining colors interchangeably. At first, this seems like a minor change, but the asymmetric constraints force us to develop fundamentally different primitives and gadgets.

2 Preliminaries

2.1 3-Coloring and Gadgets

A *proper n-coloring* of a graph $G = (V, E)$ is a function $c : V \to \{0, 1, 2, ..., n-1\}$ such that for every edge $(u, v) \in E$, we have $c(u) \neq c(v)$. The 3-coloring problem asks whether a given graph admits a proper 3-coloring.

Having defined graph coloring, we asked a natural question: *What happens when we fix some colors to certain vertices and observe how this constraint affects other vertices?* More precisely, suppose we pick some vertices as *inputs* by fixing their colors to specific values, and then examine the possible colors of a designated *output* vertex in any valid coloring scheme. This input-output behavior is similar to a machine, or a mathematical function: we provide inputs and observe the resulting output. These graph structures, which encode input-output relationships through graph coloring, are called *gadgets*. A gadget implements a function from input colors to output colors, constrained by the graph's internal structure.

Having established this framework, we ask: *Can we encode Boolean operations within this gadget system?*

2.2 Single anchor system

The answer is yes. To implement Boolean functions, our gadgets must be able to distinguish between the values TRUE and FALSE. We achieve this by introducing *anchor vertices*, vertices that are fixed to a color regardless of the input configuration, serving as reference values within this framework. Traditional constructions employ *two anchor vertices*: one fixed to a color representing TRUE, another to a color representing FALSE. This approach provides symmetric reference points, resulting in cleaner and smaller gadget configurations. However, in a theoretical sense, instead of two anchors, we actually need only one anchor to fix some logical value for distinguishing between TRUE and FALSE. We call this a *single anchor system*.

Definition 2.1 (Single Anchor System)**.** In the single anchor system for k-coloring:

1. A distinguished *anchor vertex* a_0 is fixed to color 0, representing logical falsehood.
2. The remaining colors $\{1, 2, \ldots, k-1\}$ all representing logical truth, used interchangeably.
3. For any vertex v, we interpret:
   - $v \equiv 1$ iff $c(v) \in \{1, 2, \ldots, k-1\}$
   - $v \equiv 0$ iff $c(v) = 0$

We will adopt the single anchor system for 3-coloring throughout paper unless stated otherwise.

This system is *minimal* in the sense that it uses the fewest possible fixed colors while still being able to express logical operations. However, with this asymmetry and lack of constraint on colors, gadgets become more complex and fundamentally new structures emerge.

2.3 Basic Definitions and Notation

Throughout this paper, we will work with single anchor systems, using the notation of 2.1, and let $S = \{0, 1, 2\}$ to denote the set of all colors in 3-coloring. Let $T = \{1, 2\} \subset S$ to denote the true colors in S.

We denote the behavior of a gadget as a mapping from input colors to output colors. If a gadget maps an input color i to a output color j, we write $i \to j$. If the output can be one of several colors, we write $i \to \{j_1, j_2, \ldots\}$. When two colors can map to each other interchangeably, we write $i \leftrightarrow j$.

When expressing the colors of vertices, we treat gadgets in the expression as functions maping input colors from S to sets of possible output colors according to their mapping. For example, a gadget $\mathcal{G}$ with n inputs can be viewed as the function $\mathcal{G} : S^n \to \wp(S)$.

A gadget is called *reversible* if it retains the same mapping when its input and output vertices are swapped.

3 Primitives

3.1 Definition

Primitives are small gadgets that implement basic operations. These primitives serve as building blocks for constructing more complex logical machinery within our system.

3.2 Core Primitives

3.2.1 MOV (Move)

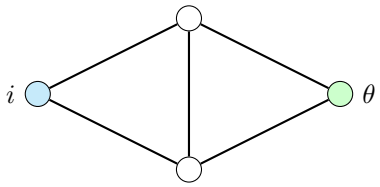


Figure 1: MOV gadget structure

The MOV gadget shown above simply moves the input color to the output. One can verify the mapping, which is only:

$$i \longrightarrow i$$

3.2.2 NOT

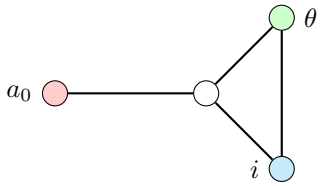


Figure 2: NOT gadget structure

The NOT gadget shown above implements Boolean NOT operation and has the following mapping:

$$0 \longleftrightarrow \{1, 2\}$$

Notably, this is the smallest gadget that's capable of implementing a Boolean operation.

3.2.3 k-NOT

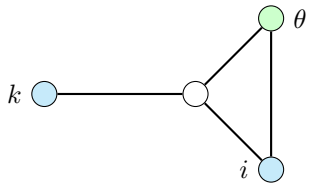


Figure 3: k-NOT gadget structure

After investigating some Boolean gates, we needed a primitive to express their behavior more concisely. Minimal gadgets have fewer vertices, which makes it easy for them to have some triangles. To address this, we generalize the NOT gadget, utilizing its triangle.

The k-NOT gadget shown above has the following mapping:

$$\begin{aligned} i = k &\longrightarrow S \setminus \{k\} \\ \text{otherwise} &\longrightarrow k \end{aligned}$$

3.2.4 ROT (Rotate)

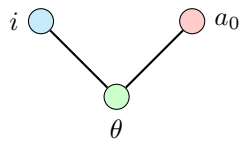


Figure 4: ROT gadget structure

The ROT gadget shown above has the following mapping:

$$0 \longrightarrow \{1, 2\}$$
$$1 \longleftrightarrow 2$$

While mapping 0 to any true colors, this gadgets swaps the true colors between themselves.

3.2.5 ROT$_s$ (Rotate states 1 and 2)

To take advantage of the asymmetric nature of the system, we tweak the idea of ROT, and develop a new gadget ROT$_s$. Unlike ROT, this gadget fixes 0 and swaps true colors between themselves.

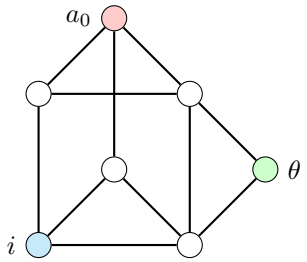


Figure 5: ROT$_s$ gadget structure

This gadget is also reversible, as one can verify the mapping after swapping the input with the output. This mapping also offers great flexibility when working with the asymmetric structure. Let us investigate its structure by decomposing it into two k-NOTs.

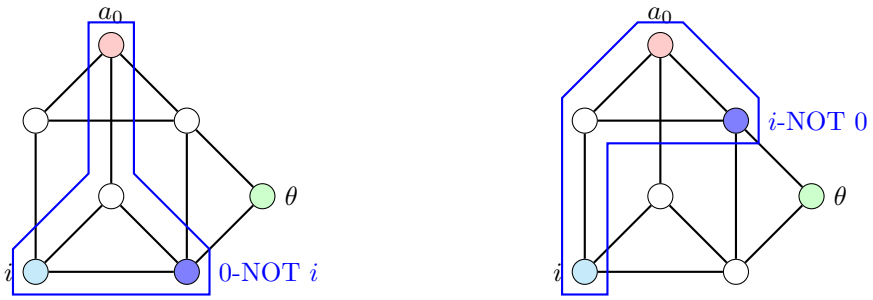


Figure 6: Decomposition of ROT$_s$ to two k-NOTs

As shown above, the ROT$_s$ can be decomposed into two k-NOTs: 0-NOT i and i-NOT 0. Thus, these constraints are sufficient to fully describe the gadget, as it can be constructed completely from them:

1. $k \in i$-NOT 0
2. $m \in$ 0-NOT $i \setminus \{k\}$
3. $\theta \in S \setminus \{k, m\}$

Theorem 3.1. *ROT_s implements the following mapping:*

$$0 \longrightarrow 0$$
$$1 \longleftrightarrow 2$$

Proof. We consider cases $i = 0$ and $i \in T$.

Case 1: $i = 0$. Then $k \in T$ and $m \in T \setminus \{k\} = \text{ROT } k$. Since $k \in T$, $k \notin \text{ROT } k \subseteq T$ hence $k \neq m$. Thus, this yields $\theta = 0$.

Case 2: $i \in T$. Then $k \in i\text{-NOT } 0 = \{i\}$ and $m \in 0\text{-NOT } i \setminus \{k\} = \{0\}$. Thus, $\theta \in S \setminus \{i, 0\}$ yields $\theta \in T \setminus \{i\} = \text{ROT } i$.

In summary, $i = 0$ corresponds to $\theta = 0$, and $i \in T$ corresponds to $\theta \in \text{ROT } i$. □

4 Ladgets

4.1 Definition

A *ladget* (a *logical gadget*) is a gadget that implements a Boolean function. Intuitively, a ladget $\mathcal{L}$ acts as a machine that takes Boolean input values $(i_1, i_2, \ldots, i_n)$ and outputs a Boolean value θ that is identical across all possible 3-colorings for the same input assignment.

Definition 4.1 (Ladget)**.** A ladget $\mathcal{L}$ is defined as follows:

Let $\mathcal{L} = (G, a_0, I, \theta)$, where $G = (V, E)$ is the graph, $a_0 \in V$ is the anchor vertex, $I = (i_1, i_2, \ldots, i_n)$ is the ordered tuple of input vertices, and $\theta \in V$ is the output vertex.

We say that $\mathcal{L}$ *implements a Boolean function* $\varphi_{\mathcal{L}} : \{0,1\}^n \to \{0,1\}$ where $\varphi_{\mathcal{L}}$ *depends on* every input, if the graph satisfies:

1. **Universality:** For every assignment of colors to the input vertices, there is a valid 3-coloring of $\mathcal{L}$.
2. **Consistency:** For every valid 3-coloring of $\mathcal{L}$, if the input vertices have Boolean values $(v_1, v_2, \ldots, v_n)$, then the output vertex θ satisfies:

$$\theta \equiv \varphi_{\mathcal{L}}(v_1, v_2, \ldots, v_n)$$

We refer to $|V|$ as the order of a ladget $\mathcal{L}$, denoted by $|\mathcal{L}|$.

Definition 4.2 (Minimality)**.** A ladget $\mathcal{L}$ is called *minimal* if there exists no other ladget implementing $\varphi_{\mathcal{L}}$ in strictly smaller order than $|\mathcal{L}|$.

Now let us investigate several minimal ladgets in detail.

4.2 Minimal Ladgets for Standard Operations

4.2.1 NAND

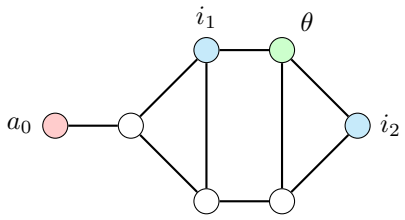


Figure 7: $\mathcal{L}_{\text{NAND}}$ ladget structure

The NAND ladget $\mathcal{L}_{\text{NAND}}$ shown above, has 7 vertices and it is one of the two minimal NAND ladgets according to our exhaustive search (Section 5.2).

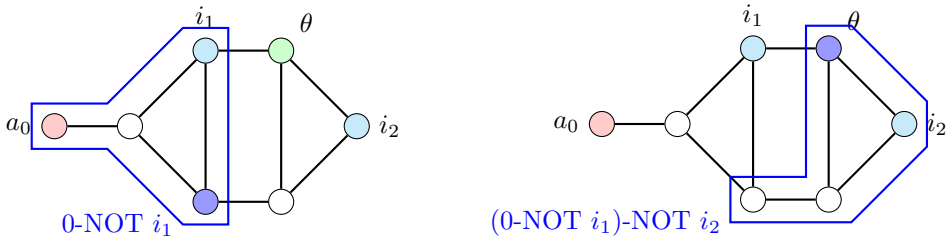


Figure 8: $\mathcal{L}_{\text{NAND}}$ decomposition

Using this decomposition, we can express θ as:

$$\theta \in (\text{0-NOT } i_1)\text{-NOT } i_2 \setminus \{i_1\}$$

Theorem 4.1. *$\mathcal{L}_{NAND}$ implements the Boolean NAND operation:*

$$(1,1) \longrightarrow 0$$
$$\textit{otherwise} \longrightarrow 1$$

Proof. We consider cases $\theta \equiv 0$ and $\theta \equiv 1$.

Case 1: $\theta \equiv 0$. This implies that $i_1 \in T$. Since 0-NOT $i_1 = \{0\}$, $\theta \in \{0\} = $ 0-NOT i_2, implying $i_2 \in T$. Thus, this yields $i_1 \equiv i_2 \equiv 1$.

Case 2: $\theta \equiv 1$. Then (0-NOT i_1)-NOT $i_2 \setminus \{i_1\} \subseteq T$. $i_1 \in T$ requires $i_2 = 0$ since 0-NOT i_1 would equal $\{0\}$. Similarly, $i_2 \in T$ requires $i_1 = 0$ since we need 0-NOT $i_1 \subseteq T$. Also $i_1 = i_2 = 0$ follows with $\theta \in T$. Thus, this covers all the remaining cases.

In summary, $\theta \equiv 0$ corresponds to $i_1 \equiv i_2 \equiv 1$, and $\theta \equiv 1$ corresponds to $i_1 \equiv 0$ or $i_2 \equiv 0$. □

4.2.2 OR

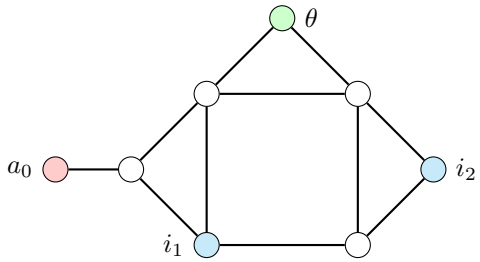


Figure 9: $\mathcal{L}_{\text{OR}}$ ladget structure

The OR ladget $\mathcal{L}_{\text{OR}}$ shown above has 8 vertices and it is one of the two minimal OR ladgets according to our exhaustive search (Section 5.2). Interestingly, this gadget resembles the NAND gate we analyzed earlier, as it only has one additional vertex and two edges, allowing us to decompose similarly using the Boolean operation NAND when investigating this graph:

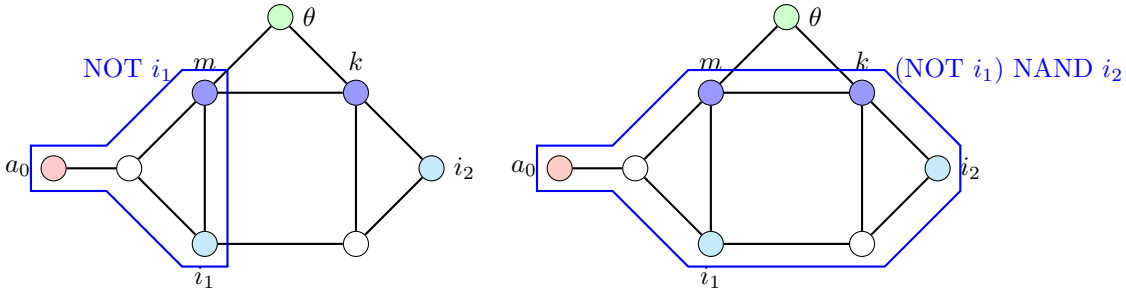


Figure 10: $\mathcal{L}_{\text{OR}}$ decomposition

Using the decomposition and labeling above, these constraints are sufficient to fully describe the gadget, as it can be constructed completely from them:

1. $m \in \text{NOT } i_1$
2. $k \in m \text{ NAND } i_2$
3. $\theta \in S \setminus \{m, k\}$

Theorem 4.2. *$\mathcal{L}_{OR}$ implements the Boolean OR operation:*

$$(0,0) \longrightarrow 0$$
$$\textit{otherwise} \longrightarrow 1$$

Proof. We consider cases $i_2 \equiv 0$ and $i_2 \equiv 1$.

Case 1: $i_2 \equiv 0$. Then we observe that the graph structure is the same as ROT_s, resulting $\theta \in \text{ROT}_s\ i_1$. Thus, this yields $\theta \equiv i_1$.

Case 2: $i_2 \equiv 1$. Then $k \in m$ NAND i_2, following logically with $k \equiv \text{NOT } m \equiv \text{NOT NOT } i_1 \equiv i_1$. Since $m \equiv \text{NOT } i_1$ and $k \equiv i_1$, therefore either $m \equiv 0$ or $k \equiv 0$. Thus implying $\theta \equiv 1$.

In summary, $\theta \equiv 0$ corresponds to $i_1 \equiv i_2 \equiv 0$, and $\theta \equiv 1$ corresponds to $i_1 \equiv 1$ or $i_2 \equiv 1$. □

4.2.3 AND

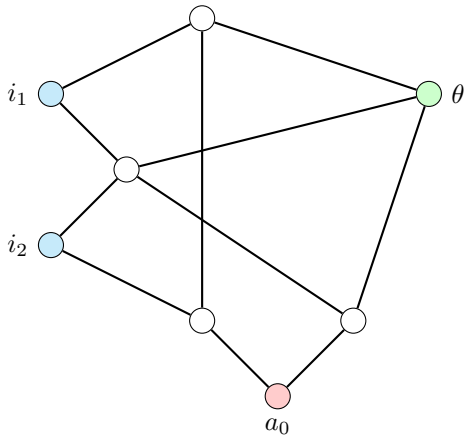


Figure 11: $\mathcal{L}_{\text{AND}}$ ladget structure

The AND ladget $\mathcal{L}_{\text{AND}}$ shown above has 8 vertices and it is one of the three minimal AND ladgets according to our exhaustive search (Section 5.2).

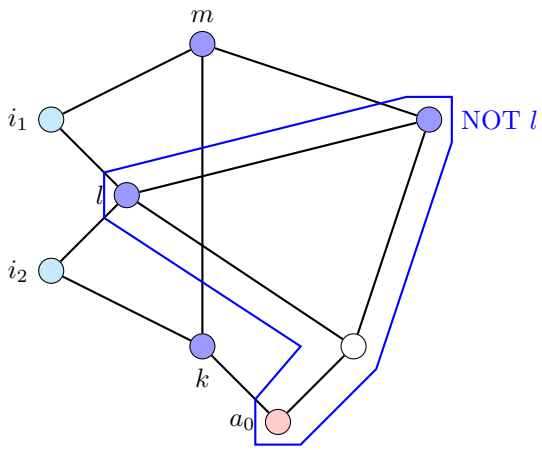


Figure 12: $\mathcal{L}_{\text{AND}}$ labeling and decomposition

Using the decomposition and labeling above, these constraints are sufficient to fully describe the gadget, as it can be constructed completely from them:

1. $l \in S \setminus \{i_1, i_2\}$
2. $k \in \text{ROT } i_2$
3. $m \in S \setminus \{i_1, k\}$
4. $\theta \in (\text{NOT } l) \setminus \{m\}$

Theorem 4.3. *$\mathcal{L}_{AND}$ implements the Boolean AND operation:*

$$(1,1) \longrightarrow 1$$
$$\textit{otherwise} \longrightarrow 0$$

Proof. We consider cases $\theta \equiv 0$ and $\theta \equiv 1$.

Case 1: $\theta \equiv 0$. Then $m \in T$ and NOT $l = \{0\}$, hence $l \in T$. Since $l \in S \setminus \{i_1, i_2\}$, this yields $i_1 \equiv 0$ or $i_2 \equiv 0$.

Case 2: $\theta \equiv 1$. Then $(\text{NOT } l) \setminus \{m\} \subseteq T$ requires $l = 0$. Then $0 \in S \setminus \{i_1, i_2\}$, therefore $i_1 \not\equiv 0 \not\equiv i_2$.

In summary, $\theta \equiv 0$ corresponds to $i_1 \equiv 0$ or $i_2 \equiv 0$, and $\theta \equiv 1$ corresponds to $i_1 \equiv i_2 \equiv 1$. □

4.2.4 XOR

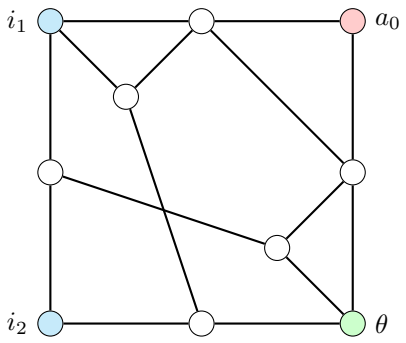


Figure 13: $\mathcal{L}_{\text{XOR}}$ ladget structure

The XOR ladget $\mathcal{L}_{\text{XOR}}$ shown above has 10 vertices and it is one of the four minimal XOR ladgets according to our exhaustive search (Section 5.2).

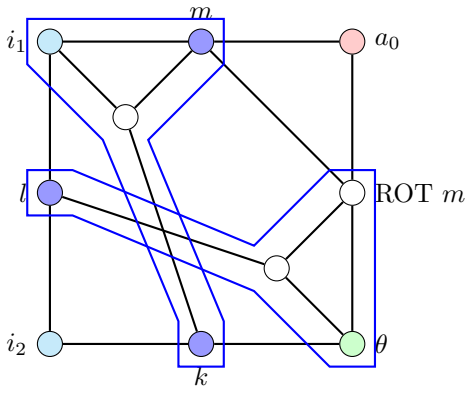


Figure 14: $\mathcal{L}_{\text{XOR}}$ labeling and decomposition

Using the decomposition and labeling above, these constraints are sufficient to fully describe the gadget, as it can be constructed completely from them:

1. $k \neq i_2$
2. $l \in S \setminus \{i_1, i_2\}$
3. $m \in k\text{-NOT } i_1 \setminus \{0\}$
4. $\theta \in l\text{-NOT}(\text{ROT } m) \setminus \{k\}$

Theorem 4.4. *$\mathcal{L}_{XOR}$ implements the Boolean XOR operation:*

$$i_1 \equiv i_2 \longrightarrow 0$$
$$i_1 \not\equiv i_2 \longrightarrow 1$$

Proof. We consider cases $\theta \equiv 0$ and $\theta \equiv 1$.

Case 1: $\theta \equiv 0$. Then l-NOT(ROT m) $\setminus \{k\} = \{0\}$. Hence $k \in T$ and either $l = 0$ or $l \in$ ROT m.

- **Subcase 1.1:** $l = 0$. Then $i_1 \neq 0 \neq i_2$ therefore $i_1 \equiv i_2 \equiv 1$.
- **Subcase 1.2:** $l \in$ ROT m. Then $i_1, i_2 \notin$ ROT m. Also $k = m$ since $m \in T$ and $\theta = l$-NOT(ROT m) $\setminus \{k\}$ should output only 0. Since $i_2 \notin$ ROT m and $m = k \neq i_2$, $i_2 = 0$. Also $k = m = m$-NOT $i_1 \setminus \{0\}$ implies $i_1 \neq m$, hence $i_1 = 0$. Thus, $i_1 \equiv i_2 \equiv 0$.

Case 2: $\theta \equiv 1$. l-NOT(ROT m) $\setminus \{k\} \subseteq T$ implies $l \in T$ since $m \in T$. Since $l \in T$, either $l \in$ ROT m or $l = m$.

- **Subcase 2.1:** $l \in$ ROT m. This implies $k = 0$ since l-NOT(ROT m) $= \{0, m\}$ but $\theta \in T$. Therefore $m \in 0\text{-NOT } i_1 \setminus \{0\}$. $0 = k \neq i_2$ and $0 \notin 0\text{-NOT } i_1$ yields $i_1 \equiv 0$ and $i_2 \equiv 1$.
- **Subcase 2.2:** $l = m$. This implies $k \neq m$ since m-NOT(ROT m) $\setminus \{m\}$ would be $\emptyset$. Also $i_1 \neq m \neq i_2$. $m \in k\text{-NOT } i_1 \setminus \{0\}$ implies $i_1 = k$, since $i_1 \neq k$ implies $m = k$ but $m \neq k$. $k \neq m$ therefore $k = 0$ or $k \in$ ROT m. Since $i_1 = k$ and $i_2 \neq k$, $k = 0$ yields $i_1 = 0$ with $i_2 \in T$ and $k \in$ ROT m yields $i_1 \in$ ROT m with $i_2 = 0$. Thus, $i_1 \equiv 0 \not\equiv i_2$ or $i_1 \not\equiv 1 \equiv i_2$.

In summary, $\theta \equiv 0$ corresponds to $i_1 \equiv i_2$, and $\theta \equiv 1$ corresponds to $i_1 \not\equiv i_2$. □

In addition, we observe that $k \neq l$ in every case above. Therefore, connecting the vertices k and l retains the universality, yielding another minimal ladget that implements XOR.

Also, we flip the 0-NOT θ as follows and obtain the ladget $\mathcal{L}_{\text{XNOR}}$ implementing XNOR:

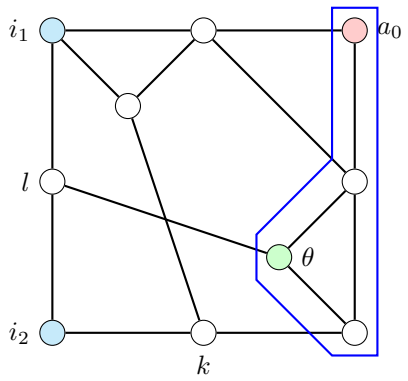


Figure 15: Obtaining $\mathcal{L}_{\text{XNOR}}$ by flipping the 0-NOT θ in $\mathcal{L}_{\text{XOR}}$

Likewise, we can also obtain another ladget that implements XNOR by connecting k and l, since $k \neq l$. These two ladgets are the only minimal ladgets that implement XNOR.

4.3 Structural Constraints

Through analysis of valid ladgets, we identified several necessary structural properties that any minimal ladget must satisfy. These constraints can be used to efficiently filter (Section 5.1.1) the ladget space while preserving the valid minimal ladgets.

Let $G = (V, E)$ be the graph of ladget $\mathcal{L}$ with designated vertices: anchor a_0, inputs $I = \{i_1, i_2, \ldots, i_n\}$, and output θ. We call vertices in the set $V \setminus (I \cup \{a_0, \theta\})$ *internal vertices.*

Theorem 4.5 (Universality Constraints)**.** *Let $\mathcal{L} = (G, a_0, I, \theta)$ be a ladget, then:*

1. *Any two inputs are non-adjacent*
2. *a_0 is not adjacent to any of the inputs*
3. *No internal vertex is adjacent to three vertices of $I \cup \{a_0\}$*

Proof. Any violation would contradict universality.

1. Let $i_j, i_k \in I$ be adjacent. Then any valid 3-coloring must satisfy $c(i_j) \neq c(i_k)$. This forbids the input assignment $i_j = i_k$, violating universality.
2. Let $i_j \in I$ and a_0 be adjacent. Then any valid 3-coloring must satisfy $c(i_j) \neq 0$. This forbids the input assignment $i_j = 0$, violating universality.
3. Let an internal vertex v be adjacent to three distinct vertices $k, l, m \in I \cup \{a_0\}$, then $c(v)$ must be different than $c(k), c(l), c(m)$.
   - If one of k, l, m is a_0, let $k = a_0$, then the assignment $l = 1, m = 2$ leaves no available color for v, violating universality.
   - If all three are inputs, the assignment $k = 0, l = 1, m = 2$ leaves no available color for v, violating universality.

□

Theorem 4.6 (Non-trivial Output)**.** *Let $\mathcal{L} = (G, a_0, I, \theta)$ be a ladget, then a_0 and θ are non-adjacent.*

Proof. Let θ and a_0 be adjacent. Then any valid 3-coloring must satisfy $c(\theta) \neq c(a_0) = 0$. Thus, regardless of the input assignment, $\theta \equiv 1$, and $\mathcal{L}$ implements a constant Boolean function $\varphi_{\mathcal{L}}(i_1, \ldots, i_n) \equiv 1$, independent of any input. □

Theorem 4.7 (Degree Constraints)**.** *Let $\mathcal{L} = (G, a_0, I, \theta)$ be a ladget, then:*

1. $\deg(\theta) \geq 2$
2. *If $\mathcal{L}$ is minimal, then for any internal vertex v,* $\deg(v) \geq 3$

Proof. We consider the two conditions separately.

1. Suppose that $\deg(\theta) = 1$ and let $v \in V$ be the only vertex adjacent to θ. Then $c(\theta) \neq c(v)$.
   - If $c(v) \in \{1, 2\}$, then θ could take more than one color not responding to a single logical value, violating the definition.

- If $c(v) = 0$, then $\theta \equiv 1$ regardless of the input assignment, contradicting the non-trivial output requirement. (Theorem 4.6)

2. Suppose that $\mathcal{L}$ is minimal and let v be an internal vertex with $\deg(v) \leq 2$. Then v has at most 2 neighbors, therefore it is possible to always find a color assignment for v, not enforcing any rules between the adjacent vertices. Thus, v is redundant and can be removed to obtain a smaller ladget $\mathcal{L}'$, contradicting the minimality of $\mathcal{L}$.

□

Theorem 4.8 (Input Degree)**.** *Let $\mathcal{L} = (G, a_0, I, \theta)$ be a ladget, then for all inputs $i_j \in I$,* $\deg(i_j) \geq 2$.

Proof. For an input $i_j \in I$, suppose that $\deg(i_j) = 1$ and $\varphi_{\mathcal{L}}$ depends on i_j. Then $v \in V$ is the only vertex adjacent to i_j. Since i_j is adjacent to only v, the color of i_j propagates through the graph with only v, therefore there are no other vertices that has any information of i_j's color that could constrain v based on i_j. Consider two different colors $c_1, c_2 \in S$:

- If $i_j = c_1$, then $v \in S \setminus \{c_1\}$
- If $i_j = c_2$, then $v \in S \setminus \{c_2\}$

Since $(S \setminus \{c_1\}) \cap (S \setminus \{c_2\}) \neq \emptyset$, there exist one color q_1 in the intersection that v can take under both input assignments. Even if v is forced to take a different color, there exist a different color q_2 under a different input assignment since v can't be constrained with 2 different colors as this would fix v to a color regardless of i_j, making $\varphi_{\mathcal{L}}$ independent of i_j. Thus, $v = q_1$ is possible in 2 input assignments, making it indistinguishable between 2 different inputs, rendering $\varphi_{\mathcal{L}}$ independent of i_j. This contradiction shows that $\deg(i_j) \geq 2$. □

5 Results

5.1 Exhaustive Search

We performed exhaustive enumeration, generating all non-isomorphic connected graphs with 7-10 vertices using the `geng` tool from the `nauty` package [2], identifying all minimal ladgets implementing NAND, AND, OR, NOR, XOR, and XNOR.

5.1.1 Verification

For a 2-input gadget with n vertices, each configuration includes:

- 1 anchor vertex
- 2 input vertices
- 1 output vertex
- $n - 4$ internal vertices

For each configuration (a_0, i_1, i_2, θ), we verified whether the graph implements a Boolean function as follows:

1. **Structural check:** Verify that the gadget configuration meets the structural requirements presented in Section 4.3.

2. **Universality check:** For each input assignment $(i_1, i_2) \in \{0, 1, 2\}^2$, verify that a valid 3-coloring exists.
3. **Consistency check:** For all valid colorings with the same input assignment, verify that the output θ has the same logical value.
4. **Truth table comparison:** Compare the resulting truth table with the target Boolean function.

In the structural check step, we were able to filter out approximately 98.8% of the gadget space with 10 vertices, reducing the overall computation time by roughly 8 times.

5.1.2 Implementation

The search was implemented in Python using NetworkX [1] with nauty's geng tool.

The verification was parallelized across 19 CPU cores, and the most computationally intensive search (10 vertices, approximately 29.5 billion configurations) required approximately 1 hour (previously 8 hours without structural checks) on Intel Core i9-13900H processor with 32 GB of memory.

5.2 Search Results

Function	Vertices	Count	Rarity (in Graphs)	Rarity (in Configs)	Rarity (filtered)
NOT	4	1	1 in 6	1 in 72	1 in 2
NAND	7	2	1 in 426	1 in 1.79×10^5	1 in 622
AND	8	3	1 in 3,705	1 in 3.1×10^6	1 in 19,417
OR	8	2	1 in 5,558	1 in 4.6×10^6	1 in 29,126
NOR	10	20	1 in 5.85×10^5	1 in 1.4×10^9	1 in 1.75×10^7
XOR	10	4	1 in 2.9×10^6	1 in 7.3×10^9	1 in 8.76×10^7
XNOR	10	2	1 in 5.8×10^6	1 in 1.47×10^{10}	1 in 1.75×10^8

Table 1: Minimal implementations found through exhaustive search

The full list of minimal implementations are available with graph6 strings in Appendix A.

Note: While geng ensures graphs are non-isomorphic, different gadget configurations within the same graph may be isomorphic to each other under graph automorphisms. For instance, if a graph has reflectional symmetry (as in the case of $\mathcal{L}_{\mathrm{XOR}}$), two configurations that are reflections of each other implement the same function. The counts reported include only non-isomorphic gadgets.

5.3 Rarity

The exhaustive search reveals that ladgets are exceptionally rare in the space of all graphs. As shown in Table 1, the probability of a random graph configuration implementing a specific Boolean function decreases dramatically with gate complexity.

The most striking case is XNOR: of the approximately 29 billion 10-vertex configurations examined, only two non-isomorphic implementations exist. This corresponds to a rarity of

1 in 14.7 billion configurations (1 in 175.2 million after filtering out gadgets not meeting the structural requirements), or 1 in 5.8 million graphs. This extreme rarity demonstrates that logical expressiveness imposes harsh constraints on gadgets.

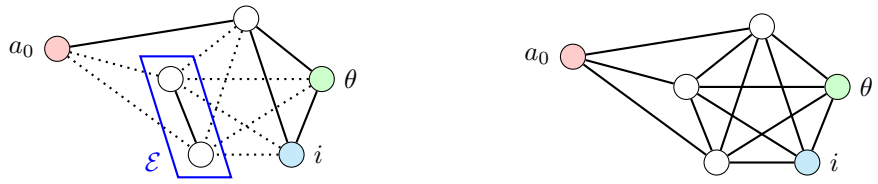
6 Embedding into k-Coloring

We present a simple technique to embed any 3-coloring ladget $\mathcal{L}$ with $|I| \leq 2$ into k-coloring while preserving its Boolean function $\varphi_{\mathcal{L}}$.

Let $\mathcal{L} = (G, a_0, I, \theta)$ be a 3-coloring ladget with $|I| \leq 2$ and graph $G = (V, E)$. We construct an *embedding package* $\mathcal{E}$, defined as the complete graph K_{k-3}. We then connect every vertex of $\mathcal{E}$ to every vertex of G, obtaining a new graph G' and then define a new k-coloring ladget $\mathcal{L}_k = (G', a_0, I, \theta)$. In any valid k-coloring, the vertices of $\mathcal{E}$ occupy $k-3$ distinct colors different from the anchor and input vertices, since all vertices in $\mathcal{E}$ are adjacent to all vertices in $I \cup \{a_0\}$; these colors are considered redundant. Since every vertex of G is adjacent to all vertices of $\mathcal{E}$, the additional $k-3$ redundant colors are forbidden on V, restricting G to a 3-coloring palette within the k-coloring scheme.

Thus, this construction embeds any 3-coloring ladget with less than 3 inputs into a valid k-coloring ladget without altering its logical behavior.

Below is an example embedding of $\mathcal{L}_{\text{NOT}}$ into 5-coloring:



(a) Embedding package $\mathcal{E}$ and $\mathcal{L}_{\text{NOT}}$ (b) After embedding

Figure 16: Embedding of $\mathcal{L}_{\text{NOT}}$ into 5-coloring

References

[1] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[2] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

Author e-mail address: `fikretgungor@hacettepe.edu.tr`

Appendices

A List of Minimal Ladget Configurations

Below is the list of minimal 3-coloring ladgets identified through our exhaustive search.

Minimal NAND implementations

Graph6	a_0	θ	i_1	i_2
`FCZeO`	3	4	2	6
`FCZUO`	2	5	1	3

Minimal OR implementations

Graph6	a_0	θ	i_1	i_2
`GCQeMo`	2	3	4	6
`G?optW`	1	2	0	3

Minimal AND implementations

Graph6	a_0	θ	i_1	i_2
`GCQbeK`	4	7	2	3
`G?q‘ug`	2	0	1	3
`GCR‘qk`	0	7	2	4

Minimal NOR implementations

Graph6	a_0	θ	i_1	i_2
I?BD?psjO	3	4	1	6
I?BD?p{lO	4	3	1	6
I?BDAo{lO	4	3	1	6
I?BDAqtN_	4	3	1	6
I?B@‘YYZ_	0	1	3	7
I?B@‘ZYj_	1	0	3	7
I?‘DAbkdo	1	4	2	3
I?‘FAqkF_	2	0	3	5
I?‘D‘pesO	2	3	4	5
I?bBD‘[s_	4	5	2	3
I?bB@qqQo	3	2	4	5
I?b@baidO	2	3	1	4
I?b@aTwy_	3	2	5	7
I?b@dpMh_	3	2	1	4
I?bERGwdG	2	3	1	4
I?‘aeIqq_	2	3	4	7
I?‘adIWoo	3	4	1	2
I?‘eKpocg	2	1	3	4
I?‘eHrWgo	3	4	1	2
I?‘cmPogg	1	2	3	4

Minimal XOR implementations

Graph6	a_0	θ	i_1	i_2
I?‘DU_[X_	2	5	0	3
I?‘DU‘eFO	2	5	0	3
I?b@dHYy?	3	2	4	5
I?b@bFWd_	2	1	4	9

Minimal XNOR implementations

Graph6	a_0	θ	i_1	i_2
I?‘DU_[X_	2	1	0	3
I?‘DU‘eFO	2	1	0	3