# Exact Topological Compliance: Generating Persistence-Equivalent Graphs

**Mattie Ji**[*]
University of Pennsylvania
mji13@sas.upenn.edu

**Indradyumna Roy**[*]
Aalto University
indradyumna.roy@aalto.fi

**Vikas Garg**
YaiYai Ltd & Aalto University
vgarg@csail.mit.edu

## Abstract

Persistent homology summarizes the topological evolution of a graph as persistence diagrams (PDs). Generating graphs whose topology matches a target descriptor is increasingly important, yet existing topology-aware generative models rely on soft regularization, producing graphs whose PDs may only be approximately similar. We initiate a principled investigation of the generation problem: produce graphs that realize a target PD exactly. We propose two approaches guided by the implicit local and global constraints a target PD imposes. The local approach builds degree-based PD-equivalent graphs iteratively, and we prove that every output realizes the target PD and that every degree-based PD-equivalent graph is reachable. However, as shown thereafter, local generation necessarily trades off between sample diversity and reaching infeasible states. We therefore recast generation as a global constraint satisfaction problem with a complete encoding solvable by modern constraint programming solvers. This is applicable to any vertex-based permutation equivariant filtration scheme and is minimal under the degree filtration. Experiments on four standard graph generation benchmarks confirm that both methods realize the target PD exactly. Together, our methods provide a principled treatment of strict topology-compliant graph generation. Code is available at `https://github.com/Aalto-QuML/ExactCompliance`.

## 1 Introduction

Topological structure is a primary driver of function in many real-world graphs. The presence of components, cycles, and higher-order voids govern stability and catalytic activity in molecules [59, 49, 55], binding affinity and virality [56, 57], information flow in brain networks [47], certain patterns in medical imaging [33], and structural risk in transportation, trade, and infrastructure networks [58, 28, 53]. Persistent homology [4, 14] is the standard tool for extracting this information. Given a graph $G$ and a filtration scheme, persistent homology tracks the births and deaths of homological features along the induced subgraph sequence. The result is a multiset of pairs $(b, d)$ called the persistence diagram (PD). PDs are stable under perturbation [8], computable with efficient algorithms [38], and have become the workhorse of topological data analysis (TDA) in machine learning, underpinning almost every topology-aware approach to graph and higher-order neural networks [19, 22, 2, 21, 24, 23, 62, 5, 44, 51, 39] and providing topological signals that standard message passing alone cannot recover (e.g., cycle detection).

In parallel, graph generation has recently emerged as a central tool across these same domains, with autoregressive [63], one-shot [46], and diffusion-based [36, 27, 52] models. Recent work conditions generation on domain-specific structural properties: spectral profile [32], molecular validity [10, 26], and structural constraints like planarity and acyclicity [30]. Conditioning on *topology* is a natural next step, but largely unexplored. A recent attempt [40], augments discrete diffusion with a 1-Wasserstein

[*]Equal Contributions.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Table 1: Overview of the core contributions in our paper.

| | |
|---|---|
| **Local Generation of PD-Equivalent Filtrations (Section 3)** | |
| Local Generation for General/Vertex/Degree Filtrations | Algorithm 1, 2, 3 |
| Correctness and Diversity of Local Generations | Theorem 3.1, 3.7, 3.9 |
| Perfect Generation on Acyclic Graphs (for General/Vertex) | Corollary 3.4, 3.8 |
| Impossibility Results for Local Generation | Theorem 3.10 |
| **Global Generation of PD-Equivalent Graphs (Section 4)** | |
| Global Generation for Any Canonical Vertex Filtration Scheme $\Phi$ | Section 4.1 |
| Soundness, Completeness, and Degree Minimality of Encoding | Theorem 4.2 |
| **Experiments (Section 5)** | |
| Local vs. Global on Four Graph Generation Benchmarks | Table 2 |
| Comparison Against PD-Conditioned Neural Diffusion Baselines | Table 3, Figure 3 |

PD-matching loss. This soft regularizer nudges the learned distribution toward topological plausibility, but offers no guarantees that individually generated graphs attain the desired topological structure (e.g., the target number of cycles and components).

**PD-conditioned graph generation from first principles.** A PD is a finite summary of an underlying combinatorial structure and therefore collapses many graphs to the same image. The most widely studied filtration scheme in the graph learning literature is the vertex-degree function [2, 21, 40, 62, 19, 20]: it is permutation-equivariant and aligns naturally with the local message-passing inductive bias of GNNs. We adopt it as the primary filtration scheme throughout this paper and discuss extensions to other vertex-based, permutation-equivariant schemes (centrality [29], square clustering coefficients [25], triangle count [48], etc.) where they apply.

Even under this canonical choice, a PD is far from a complete descriptor. Figure 1 shows two non-isomorphic graphs with identical degree-based PDs; in general, graphs with differing structures and connectivity patterns, can share the same PD under a given filtration scheme. Each PD therefore identifies a *class of compatible graphs*, and conditioning on a PD amounts to sampling from this class. This raises the natural question: *given an input PD and filtration scheme $\Phi$, can we generate strictly PD-compatible graphs under $\Phi$?*

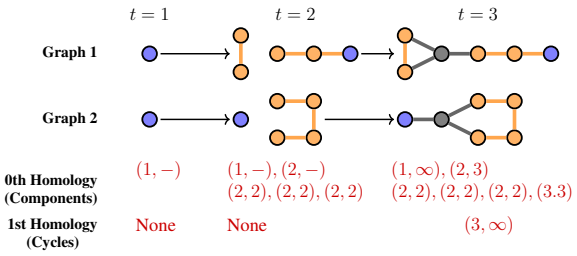



Figure 1: Two non-isomorphic graphs with identical degree-based persistence. Vertex color encodes degree; edges inherit the color of their max-degree endpoint.

This motivates a precise criterion for any PD-conditioned generator. Given a filtration scheme $\Phi$ and a graph $G$, we let $\mathcal{P}(G^{\Phi}_{\bullet})$ denote the PDs associated to the induced subgraph sequence of $G$ under $\Phi$. We call a generator ***strictly PD-compliant*** if it satisfies

1. **Soundness.** Every sample $G$ from the generator satisfies $\mathcal{P}(G^{\Phi}_{\bullet}) = D$ exactly.
2. **Completeness.** Every PD-compatible graph can be created by the generator.

Soft-conditioned models violate soundness by construction. Aiming for strict PD-compliance is the natural starting point: it makes the conditioning problem precise, and exposes what an input PD does and does not pin down. Once strict compliance is attained, it can be relaxed in applications.

**Why exact compliance?** Exact generation produces families of graphs with identical persistence diagrams but different combinatorial properties. Such families provide controlled datasets and benchmarks for training and evaluation in topological learning, and one can test whether a model truly exploits the prescribed topology or leans on unrelated statistics. They also give clean training and evaluation data for graph generation and reasoning from topological descriptors, with no label noise from approximate PD matching.

Exact generation can also help find counterexamples to conjectures about what properties are determined by persistence, and approximate generation cannot serve this purpose at all. Exactness is also important when applications critically depend on the exact number of topological features, such as ring counts in molecular generation [43] and coverage holes in sensor networks [11].

**Contributions.** We initiate a principled investigation of strict PD-compliant graph generation; Table 1 summarizes the formal statements. Our contributions are threefold.

**(1) Local iterative sampler (Section 3).** Local decisions are a prerequisite for any globally correct generation and rely only on the neighborhood-level information standard GNN pipelines already provide, making them the natural starting point. We give a sampler that is sound and complete on general, vertex-based, and degree filtrations, but the sampler can run into infeasible states. We subsequently prove an impossibility result: any local approach must trade off between locally PD-compliant choices and reaching globally infeasible states.

**(2) Global generation via constraint programming (Section 4).** The impossibility motivates us to characterize the global constraints. We recast generation as a constraint satisfaction problem applicable to any vertex-based permutation-equivariant filtration scheme, and give concrete encodings for degree, triangle and clique counts, clustering, closeness, Katz, and PageRank. We prove our method is strictly PD-compliant, and is a minimal encoding for degree filtrations.

**(3) Empirical validation (Section 5).** On COMM-20, PLANAR, SBM, and ENZYMES, both methods realise the target degree-PD exactly, while neural diffusion baselines with PD-conditioning regularisers leave a structural approximation gap that does not close with more training.

**Related Works.** Persistence-based reconstructions have been studied for embedded graphs, geometric shapes, and metric-measure spaces [7, 6, 1, 3, 45, 35, 17, 37]. These approaches exploit extrinsic information such as coordinates, directions, distances, or sampling measures to recover or analyze a geometric realization. In contrast, we consider abstract graphs without prescribed embeddings and characterize the set of graphs realizing specified persistence diagrams under an intrinsic, permutation-equivariant filtration. Our framework therefore complements geometric reconstruction methods by studying graphs sharing a common intrinsic PH signature. We only consider graph filtrations, as this is quite common for the usage of persistent homology in the GNN literature [19, 21, 22, 64, 48].

To compare the two, the persistent homology transform (PHT) [50] is injective [16, 9] on embedded graphs, but it does not resolve the graph isomorphism problem, as two different embeddings of the same abstract graph can have different PHTs. Our setting contains no coordinates, and we want to reconstruct abstract graphs with equivalent PDs, with the goal of benchmarking for graph isomorphisms. Another deviation from typical TDA considerations is that the GNN literature typically requires permutation equivariance for their filtrations. This imposes a strong rigidity on filtrations that typically do not show up for embedded graphs. For example, any permutation equivariant function on a cycle graph has to be constant, but certainly we can look at the height filtration on embedded cycles.

## 2 Background and Preliminaries

We assume a graph $G$ with vertex set $V(G)$ and edge set $E(G)$ is finite, simple, and undirected.

**Definition 2.1.** Let $G$ be a graph, a (general) **filtration** of $G$ is a sequence of subgraphs of $G$ as

$$G_\bullet : \emptyset = G_{-1} \subseteq G_0 \subseteq ... \subseteq G_n = G.$$

Note here we do allow the $G_i = G_{i+1}$ for some $i$'s. We say a filtration of $G$ is **vertex-based** if each $G_i$ is an induced subgraph of $G$ on its vertex set $V(G_i) \subseteq V(G)$. We say a vertex based filtration of $G$ is **degree-based** if $V(G_i) = \{v \in V(G) \mid \deg(v) \leq i\}$ and $n$ is the maximum degree in $G$. We write the degree-based filtration as $G_\bullet^{\deg}$.

Given any filtration $G_\bullet$ of $G$, there are lists of persistence pairs tracks the evolution of the births and deaths of its topological features on its (0-dimensional) vertex-features and (1-dimensional) cycle-features. The lists are constructed as follows.

1. For each $b \geq 0$, we can view $G_{b-1} \subseteq G_b$ as a sequence of adding vertices first and then adding edges to $G_b$. Any vertex $v$ added is said to be **born at time** $b$ and is thought of as a vertex representing the connected component that is $\{v\}$.

2. Whenever an edge is added between two components with vertex representatives $v$ and $w$ respectively in the stage $G_{d-1} \subseteq G_d$. We kill off whichever vertex in $v$ and $w$ that is born later, assign its **death time** to $d$, and assign the other vertex as the representative of the new merged component. If $v$ and $w$ were born at the same time, we simply choose one of them to kill off. A vertex representing a component alive at $G_n = G$ is assigned to die at time $+\infty$.
3. Whenever an edge is added from the same component to itself in the stage $G_{b-1} \subseteq G_b$, this is an **independent cycle**, and we assign a tuple of the form $(b, \infty)$ to it.

**Definition 2.2.** Let $G_\bullet$ be a filtration of $G$, its **persistence diagram** $\mathcal{P}(G_\bullet)$ is a pair of two **unlabeled multisets** $\mathcal{P}_0(G_\bullet)$ and $\mathcal{P}_1(G_\bullet)$. Here, $\mathcal{P}_0(G_\bullet)$, the 0-dimensional persistence diagram, is the list of birth and death time $(b, d)$ for each vertex $v \in V(G)$. $\mathcal{P}_1(G_\bullet)$, the 1-dimensional persistence diagram, is the list of cycle birth times $(b, \infty)$ for each independent cycle in $G$.

Note the lists $\mathcal{P}_0(G_\bullet), \mathcal{P}_1(G_\bullet)$ are independent of the choices made in the construction, and the final output does not identify persistence pairs with simplices of the graph. Definition 2.2 is a special case of taking persistence diagrams on simplicial complexes (see [13]). Here we restricted to the case of filtrations on graphs where the only non-empty PDs are in degree 0 and 1.

**Remark 2.3.** Some sources [2, 22] alternatively define a filtration of $G$ to come from a function $f : V(G) \cup E(G) \to \mathbb{R}$ such that for any edge $e = (v, w)$, $f(e) \geq \max(f(v), f(w))$. The persistence pairs of any $f$ is equivalent to that of Definition 2.2 by renormalizing $f$ to land in $\mathbb{Z}_{\geq 0}$. This is possible as $f$ can only take finitely many values, and the expressive power of persistence pairs only depend on the relative ordering of its values on the simplicies. Being vertex-based is equivalent to having $f(e) = \max(f(v), f(w))$. Note the degree function is vertex-based and already normalized.

In general the persistence pairs depend on the filtration. When such filtration is canonically defined (e.g., degree-based), it is a graph isomorphism invariant. Here, "canonically defined" refers to any permutation equivariant filtration scheme $\Phi$ (see Section 3 of [2]). For our purposes, we define:

**Definition 2.4.** A **filtration scheme** $\Phi$ assigns every graph $G$ a filtration function $\Phi_G : V(G) \cup E(G) \to \mathbb{R}$. $\Phi$ is **permutation equivariant** if for any graph isomorphism $\psi : G \to H$, $\phi_G = \phi_H \circ \psi$. For any such $\Phi$ valued in $\mathbb{Z}_{\geq 0}$, we write $H_\bullet^\Phi$ to denote the sequence of subgraphs associated to $\Phi$.

**Proposition 2.5** (Proposition 1 of [2])**.** A permutation equivariant filtration scheme gives the same lists of persistence diagrams on isomorphic graphs.

In this work, we are interested in generating graph filtrations whose persistence pairs match a given PH signature. For convenience, we will use the following two terminologies.

**Definition 2.6.** We say two filtrations $G_\bullet$ and $H_\bullet$ are **PD-equivalent** (persistence-diagram-equivalent) if $\mathcal{P}(G_\bullet) = \mathcal{P}(H_\bullet)$. We say $G$ and $H$ are **degree-PD-equivalent** if $\mathcal{P}(G_\bullet^{\deg}) = \mathcal{P}(H_\bullet^{\deg})$. The **$k$-th truncation** $\mathcal{P}^k(G_\bullet)$ of a filtration $G_\bullet$ is $\mathcal{P}(G_\bullet^k := G_{-1} \subseteq G_0 \subseteq ... \subseteq G_k)$.

## 3 Local Generation

In this section, we describe a local generation process for producing PD-equivalent random filtrations. We then modify the process to adapt to vertex-based and degree-based filtrations. The methods here take a **local approach** to generation by iteratively generating a sequence of subgraphs $H_\bullet$ such that their $k$-th truncations (1) $\mathcal{P}^k(H_\bullet) = \mathcal{P}^k(G_\bullet)$ agree for each $k \in \{-1, 0, ..., n\}$ and (2) the construction of $H_k$ should only depend on $H_{k-1}$ and PDs present in $G_\bullet^k$. The algorithms can generate all possible PD-equivalent filtrations if they terminate, but they can run into infeasible states that will be explained later in this section.

### 3.1 Producing PD-Equivalent General Filtrations

Here, we describe a local generation process to iteratively create a general filtration $H_\bullet$ such that $\mathcal{P}(H_\bullet) = \mathcal{P}(G_\bullet)$. The exact details are outlined in Algorithm 1 (see Appendix E). Since we aim to make $H_\bullet$ such that its truncations agrees with that of the hidden input $G_\bullet$, we describe a procedure to inductively build $H_k$ from a given $H_{k-1}$ whose previous truncations agree with that of $G_\bullet$.

1. (**Vertex Addition**): For each $(k, d) \in \mathcal{P}_0(G_\bullet)$, add a new vertex to $H_k$ with birth time $k$.
2. (**Component Merging**): We traverse through the list of tuples in $\mathcal{P}_0(G)$ that die at time $k$ in **random order**. For each tuple $(b, k)$, we pick a **random component** $X$ with birth time $b$ and

a **random component** $Y \neq X$ with birth time $\leq b$ and add an edge between a random pair of nodes in $X \times Y$. We then update the component birth times appropriately.
3. (**Cycle Addition**): For each $(k, \infty) \in \mathcal{P}_1(G_\bullet)$, add an edge between a pair $(p, q) \in X \times X$ for any component $X$ such that $p, q$ does not already share an edge.

Note here it is important to traverse in random order during the component merging step. Otherwise, there are PD-equivalent filtrations to $G_\bullet$ that Algorithm 1 could not make.

**Theorem 3.1.** Given $(\mathcal{P}_0, \mathcal{P}_1)$ the persistence pairs coming from a hidden filtration $G_\bullet$. If Algorithm 1 terminates, the filtration $H_\bullet$ is PD-equivalent to $G_\bullet$. Furthermore, any filtration $H'_\bullet$ that is PD-equivalent to $G_\bullet$ can be created by Algorithm 1.

The statement of Theorem 3.1 indicates that Algorithm 1 may not be able to terminate sometimes. It turns out Algorithm 1 may run out of cycles to add in the cycle addition stage.

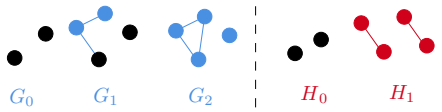



Figure 2: An example of how Algorithm 1 can run out of cycles in the cycle adding stage.

**Example 3.2.** Let $G_\bullet$ be the hidden filtration given in Figure 2. It is perfectly possible for Algorithm 1 to create $H_0 \subseteq H_1$, as $\mathcal{P}(\emptyset \subseteq H_0) = \mathcal{P}^0(G_\bullet)$ and $\mathcal{P}(\emptyset \subseteq H_0 \subseteq H_1) = \mathcal{P}^1(G_\bullet)$. However, when going from $H_1$ to $H_2$, one cannot add a single edge to run out of cycles.

We remark that cycle addition would not be an obstruction if we allow multi-edges between nodes. For convenience, we derive a simple criterion for when cycle addition would succeed. For a graph $B$, we define its **cycle capacity** as $\mathrm{cap}^{\mathrm{cyc}}(B) := \sum_{A, \text{ component of } B} \frac{|V(A)|(|V(A)|-1)}{2} - |E(A)|$.

**Proposition 3.3.** Suppose $H_{k-1}$ has been iteratively built. Algorithm 1 can always succeed in the vertex addition and component merging step. Furthermore, if the number of the tuples $(k, \infty)$ is bounded by $\mathrm{cap}^{\mathrm{cyc}}(H_{k-1})$, then Algorithm 1 can succeed in the cycle addition step.

In particular, $\mathrm{cap}^{\mathrm{cyc}}(B)$ is always non-negative, so we have as a corollary that Algorithm 1 always achieves the desired generation on acyclic graphs.

**Corollary 3.4.** Algorithm 1 always terminates if $\mathcal{P}_1(G_\bullet)$ is empty (i.e., $G$ is acyclic).

In practice, we could run Algorithm 1 repeatedly and sample from successful outputs (see Section 5.1).

**Remark 3.5.** Although we focused on vertex-based filtration, there are also **edge-based filtrations**. A filtration is edge-based if and only if all tuples in $\mathcal{P}_0$ have birth time 0. Thus, Algorithm 1 (Appendix E) can generate all possible edge-based filtrations matching the same PD.

### 3.2 Producing PD-Equivalent Vertex-Based Filtrations

Given the PH signatures $\mathcal{P}(G_\bullet)$ where $G_\bullet$ is a hidden vertex-based filtration, we describe a local generation process to iteratively create a PD-equivalent vertex-based filtration $H_\bullet$. Algorithm 2 (see Appendix E for more details) is a modification of Algorithm 1 where:

1. (**Vertex Addition**): For each new vertex added, we record them in a list $N_k$.
2. (**Component Merging**): The pair of nodes $(x, y)$ selected has $\{x, y\} \cap N_k \neq \emptyset$.
3. (**Cycle Addition**): The pair $(p, q)$ selected to make the cycle has $\{p, q\} \cap N_k \neq \emptyset$.

Modifications 1-3 above would cause obstructions during component merging. To remedy this, we introduce an **additional change** to the **component merging step**, motivated by the following.

**Proposition 3.6.** In a vertex-based filtration, any vertex death at time $k$ forces a trivial pair $(k, k)$.

(**Minimum Vertex Pivot**): If there are no deaths, then ignore this step. Otherwise, let $b_{\min} := \inf_{\text{component deaths of the form}(b,k)} b$. If the number of component whose birth time is $b_{\min}$ is equal to the number of tuples of the form $(b_{\min}, k)$, we draw an edge from a random $v \in N_k$ to a random vertex in a random component whose birth time is $< b_{\min}$. Otherwise, we draw an edge from a random $v \in N_k$ to a random vertex in a random component whose birth time is $\leq b_{\min}$.

There are still obstructions for cycle-addition, but most of preceding discussions in Section 3.1 apply.

**Theorem 3.7.** Given $(\mathcal{P}_0, \mathcal{P}_1)$ from a hidden vertex-based filtration $G_\bullet$. If Algorithm 2 terminates, the filtration $H_\bullet$ is vertex-based and PD-equivalent to $G_\bullet$. Furthermore, any vertex-based filtration

$H'_\bullet$ that is PD-equivalent to $G_\bullet$ can be created by Algorithm 2. Furthermore, Algorithm 2 can always succeed in the stages for vertex addition, minimum vertex pivot, and component merging.

**Corollary 3.8.** Algorithm 2 always terminates if $\mathcal{P}_1(G_\bullet)$ is empty.

### 3.3 Producing PD-Equivalent Degree Filtrations

Given the PH signature $\mathcal{P}(G_\bullet^{\deg})$, we now describe a modification from Algorithm 2 that generates a PD-equivalent degree-based filtration (see Algorithm 3 in Appendix E for exact details), with:

1. (**Vertex Addition**): For each new vertex $v$ added, we assign a capacity number $\mathrm{cap}(v) = k$.
2. (**Minimum Vertex Pivot**): We add the additional constraint that the edge must be drawn from $v \in N_k$ to a vertex whose capacity is positive. We subtract 1 off the capacity of the two nodes.
3. (**Component Merging**): We ensure the pair of nodes $(x, y)$ selected has $\mathrm{cap}(x), \mathrm{cap}(y) > 0$. We then subtract 1 off the capacity of the two nodes.
4. (**Cycle Addition**): We ensure the pair $(p, q)$ selected has $\mathrm{cap}(p), \mathrm{cap}(q) > 0$. We then subtract 1 off the capacity of the two nodes.

**Theorem 3.9.** Given $(\mathcal{P}_0, \mathcal{P}_1)$ the degree-based persistence coming from a hidden graph $G_\bullet^{\deg}$. If Algorithm 3 terminates, the filtration $H_\bullet$ is degree-based and has the same PDs as $G_\bullet^{\deg}$. Furthermore, any degree-based filtration $(H')_\bullet^{\deg}$ that is PD-equivalent to $G_\bullet^{\deg}$ can be created by Algorithm 3.

### 3.4 Fundamental Limitations to the Local Approach

One might wonder if the inability to overcome the cycle addition stage is because the local algorithms may not have been optimized enough. In our local approach, we seek to construct our graph filtration iteratively to match the $k$-th truncation of the given $\mathcal{P}(G_\bullet)$ such that the construction up to $H_k$ should only depend on $\mathcal{P}^k(G_\bullet)$. We show below that these obstructions are instead a fundamental limitation of any similar local approach to generation.

**Theorem 3.10.** There is no Algorithm $A$ whose input is $\mathcal{P}(G_\bullet)$, ranging over all general (resp. vertex-based, degree-based) filtrations of all graphs, and output is a (resp. vertex-based, degree-based) filtration $H_\bullet$ such that

1. Let $\mathcal{O}(G_\bullet)$ be the set of possible outputs Algorithm $A$ can make from a fixed $\mathcal{P}(G_\bullet)$. $\mathcal{O}(G_\bullet)$ is equal to the set of all general (resp. vertex-based, degree-based) $H_\bullet$ PD-equivalent to $G_\bullet$.
2. (**Local Approach**) Let $\mathcal{O}_k(G_\bullet)$ denote the set of possible $H_{-1} \subseteq ... \subseteq H_k$ that Algorithm A can make from a given $\mathcal{P}(G_\bullet)$. $\mathcal{O}_k(G_\bullet) = \mathcal{O}(G_{-1} \subseteq ... \subseteq G_k)$.

Conceptually, Theorem 3.10 is saying that trying to ensure perfect generation at each step of the filtration (i.e., proceeding locally) would miss the global structure of the PH signature. The idea is that conditions (1) and (2) forces every graph filtration PD-equivalent to a truncation of $G_\bullet$ to admit an extension to a full PD-equivalent filtration. However, we can construct examples where no such extensions exist for general, vertex-based, and degree-based filtrations.

## 4 Global Generation via Constraint Programming Formulation

Here, we take a global approach to generating PD-equivalent degree filtrations. In fact, our model is applicable to all permutation equivariant filtration scheme $\Phi$, which for simplicity of presentation we assume is valued in $\mathbb{Z}_{\geq 0}$ (this reduction is possible due to Remark 2.3).

In the local approach, one constructs a solution incrementally and makes a sequence of local decisions. This introduces a strong path dependence, which Theorem 3.10 shows are fundamentally limited. As persistent homology is inherently a global descriptor, we seek a way to fully solve the system as a single object and identify the entire feasible set of combinatorial constraints before any sampling occurs. This approach avoids the accumulation of local choices and enforces correctness on the global level. Solving this naturally leads to the usage of **constraint programming** (CP).

In this section, we recast strict PD-compliant generation as a constraint satisfaction problem over Boolean and integer variables. When $\Phi = \deg$, the encoding is *minimal-sufficient*: its satisfying

assignments are in bijection with graphs $H$ satisfying $\mathcal{P}(H_\bullet^{\deg}) = (\mathcal{P}_0, \mathcal{P}_1)$, and removing any single constraint family admits assignments that are not. Since our methodology applies to any permutation equivariant filtration $\Phi$, we also discuss how to implement various filtration functions in the solver.

### 4.1 The Encoding Implementation

Let $\Phi$ be a vertex-based permutation equivariant filtration scheme (e.g., degree) valued in $\mathbb{Z}_{\geq 0}$. Given an input $(\mathcal{P}_0, \mathcal{P}_1)$ coming from the PDs of $\Phi$ on a hidden graph $G$, we seek to generate a graph $H$ whose associated PDs under $\Phi$ matches the input.

**Pre-computed quantities.** From $(\mathcal{P}_0, \mathcal{P}_1)$ we read off: the vertex set $V = [n]$ where $n = |\mathcal{P}_0|$; the filtration value $a_i$, equal to the birth time of $i \in V$ in $\mathcal{P}_0$; the maximum filtration stage $K = \max_i a_i$; the active vertex set at stage $k$, $V_k = \{i : a_i \leq k\}$; the alive-component count $cc_k = |\{(b,d) \in \mathcal{P}_0 : b \leq k < d\}|$; the cumulative cycle count $\mathrm{cy}_k = |\{(b,\infty) \in \mathcal{P}_1 : b \leq k\}|$; and the per-birth survival count $\alpha(k,b) = |\{(b,d) \in \mathcal{P}_0 : d > k\}|$.

**Note:** Since the underlying graph $G$ is hidden, we do not know which vertex is assigned to birth/death pair in $\mathcal{P}_0$. However, this does not matter in the generation of the new graph, and we assign each $i$ to the multiset of birth/death pairs in an arbitrary bijection.

**Variables.** We model the graph to generate as the lower triangular part of an adjacency matrix $M$ with entries $M_{ij}$. To keep track of the persistence data, at each stage $k$, we also group the active vertices $V_k$ partition into $cc_k$ slots indexed by $[cc_k] = \{0, 1, \ldots, cc_k - 1\}$. Each slot is a model for a component of the graph at stage $k$ and can be thought of as the induced subgraph on vertices in the slot. We use the following variables to be solved in constraint programming:

| Variable | Domain | Semantics |
|---|---|---|
| $M_{ij}, 1 \leq i < j \leq n$ | $\{0,1\}$ | edge $\{i,j\} \in E(H)$ |
| $x_{kcv}, c \in [cc_k],\ v \in V_k$ | $\{0,1\}$ | vertex $v$ in slot $c$ at stage $k$ |
| $\pi_{kcij}, c \in [cc_k],\ i,j \in V_k,\ i \neq j$ | $\{0,1\}$ | $i$ is parent of $j$ in spanning tree of slot $c$ at stage $k$ |
| $\ell_{kcv}, c \in [cc_k],\ v \in V_k$ | $\{0,\ldots,n\}$ | depth of $v$ from the root of slot $c$ at stage $k$ |
| $r_{kcv} \coloneqq x_{kcv} \wedge \bigwedge_{u<v} \neg x_{kcu}$ | $\{0,1\}$ | Is $v$ the vertex with smallest index in slot $c$ at stage $k$? |
| $e_{kcb} \coloneqq \big(\exists\, v \in \mathrm{slot}_c^k : a_v = b\big)$ $\wedge \big(\nexists\, v \in \mathrm{slot}_c^k : a_v < b\big)$ | $\{0,1\}$ | slot $c$ at stage $k$ was born at time $b$ |

Here the variables $\pi_\bullet, \ell_\bullet$ are introduced to optimize the constraint solver, see Remark 4.1.

**Constraints.** The encoding consists of six constraint families.

$\mathcal{C}_1$ **($\Phi$ Alignment).** Each vertex aligns to its prescribed value under $\Phi$.

$$\Phi_M(i) = a_i, \qquad \forall\, i \in [n]$$

In the special case when $\Phi$ is degree, the constraint becomes:

$\mathcal{C}_1^{\deg}$ **(degree).** Each vertex realizes its prescribed degree.

$$\sum_{j \neq i} M_{ij} = a_i \qquad \forall\, i \in [n]$$

$\mathcal{C}_2$ **(partition).** At each stage, every active vertex belongs to exactly one slot.

$$\sum_{c \in [cc_k]} x_{kcv} = 1 \qquad \forall\, k,\ v \in V_k$$

$\mathcal{C}_3$ **(edge-slot consistency).** The endpoints of every edge share a slot at every stage where both are active (i.e., there is no edge between two vertices at different slots).

$$M_{uv} \ \leq\ 1 - |\, x_{kcu} - x_{kcv}\,| \qquad \forall\, k,\ c \in [cc_k],\ u < v \in V_k$$

$\mathcal{C}_4$ **(slot connectivity).** Each slot induces a connected subgraph at every stage. We witness this by forcing the existence of a spanning tree on the slot, encoded via parent pointers $\pi$ rooted at the

canonical root $r_{kcv}$, with depths $\ell$ ruling out cycles. The tree must use only graph edges within the slot ($\pi_{kcij} \le M_{ij} \wedge x_{kci} \wedge x_{kcj}$); each non-root member has exactly one parent ($\sum_{u \neq v} \pi_{kcuv} = x_{kcv} - r_{kcv}$); and depths increase along parent edges ($\pi_{kcij} = 1 \Rightarrow \ell_{kcj} = \ell_{kci} + 1$). Note that the spanning tree here is to ensure connectivity and is not used for the final graph ($M_{ij}$'s) produced.

$\mathcal{C}_5$ **(Euler characteristics).** Edge counts at each stage match the prescribed cycle-birth counts via $|E_k| - |V_k| + cc_k = \mathrm{cy}_k$.

$$\sum_{i<j,\ i,j \in V_k} M_{ij} \ - \ |V_k| \ + \ cc_k \ = \ \mathrm{cy}_k \qquad \forall\, k.$$

$\mathcal{C}_6$ **(birth-time survival).** The number of slots whose oldest member was born at $b$ equals $\alpha(k,b)$, the number of $\mathcal{P}_0$ pairs born at $b$ still alive at stage $k$.

$$\sum_{c \in [cc_k]} \mathbf{1}[\, e_{kcb}\,] = \alpha(k,b) \qquad \forall\, k,\ b,$$

where $e_{kcb}$ holds iff slot $c$ contains a vertex born at $b$ but no vertex born earlier. Note that when $\Phi$ is degree, we note that $e_{kcb} = \big(\exists\, v \in \mathrm{slot}_c^k : \deg(v) = b\big) \wedge \big(\nexists\, v \in \mathrm{slot}_c^k : \deg(v) < b\big)$.

**Remark 4.1.** We used spanning trees to enforce connectivity in $\mathcal{C}_4$ as an optimization step for the solver. A naive approach to checking connectivity would be to ensure that any two vertices in a slot has a path between them, which is about quadratic in time. Checking a spanning tree is about linear.

**Theorem 4.2** (Correctness of the encoding)**.** Let $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_6\}$, $\mathcal{F}$ the set of satisfying assignments to $\mathcal{C}$, and $\mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$ the set of graphs $H$ with $\mathcal{P}(H_\bullet^\Phi) = (\mathcal{P}_0, \mathcal{P}_1)$. Define $H(\sigma)$ from $\sigma \in \mathcal{F}$ by $E(H(\sigma)) = \{\{i,j\} : \sigma(M_{ij}) = 1\}$. Then:

1. **Soundness:** $H(\sigma) \in \mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$ for every $\sigma \in \mathcal{F}$.
2. **Completeness:** every $H \in \mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$ admits at least one $\sigma \in \mathcal{F}$ with $H(\sigma) = H$.
3. **Minimality for Degree:** When $\Phi = \deg$, for each $i \in \{1, \ldots, 6\}$, there exist a target $(\mathcal{P}_0, \mathcal{P}_1)$ and an assignment $\sigma$ that satisfies $\mathcal{C} \setminus \{\mathcal{C}_i\}$ but with $H(\sigma) \notin \mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$.

Here, soundness follows as all constraints imposed are formal properties satisfied by PDs, and completeness follows from carefully arguing these constraints completely determine the representatives. Note that the completeness of Theorem F.6 implies for any graph $G$ the solver can generate, it can also reach every representative in the same graph isomorphism class.

The full proof of Theorem 4.2 (1-2) is in Appendix F. In Appendix G, we generated 6 assignments see (Figure 4) as counter-examples in the proof for the minimality of the encoding when $\Phi = \deg$. In Appendix I, we also provide a complexity and scaling analysis of the global encoder for degree.

### 4.2 Implementation of Various Filtration Schemes

To demonstrate that the framework is not limited to degree, we implement other filtration schemes. Since only $\mathcal{C}_1$ (the alignment $\Phi_M(i) = a_i$) depends on the filtration, we specify it for each below. Recall $M_{ij}$ is the $(i,j)$-th entry in the adjacency matrix and $d_i = \sum_{j \neq i} M_{ij}$.

**(a) triangle and $k$-clique count.** For each vertex $v$, its triangle count is the number of 3-cycles on $G$ that includes the vertex $v$. We can encode this as

$$\textbf{Triangle Count:} \sum_{j<l,\ j,l \neq i} M_{ij} M_{il} M_{jl} = a_i \qquad \forall\, i \in [n].$$

Similar formulas also hold for $k$-clique count for $k \ge 3$.

**(b) clustering coefficient.** The clustering coefficient of a vertex $i$ is twice its triangle count $t_i$ divided by $d_i(d_i - 1)$ when $d_i \ge 2$, and 0 otherwise. We can encode this as the equality

$$\textbf{Clustering Coefficient:}\ 2 \sum_{j<l,\ j,l \neq i} M_{ij} M_{il} M_{jl} = a_i \max(1, d_i(d_i - 1)) \qquad \forall\, i \in [n].$$

**Remark 4.3.** As $a_i$ is rational but the solver is integer-based, we in practice scale by a positive integer $D$ to give the equation $2D\, t_i = p_i \max(1, d_i(d_i - 1))$ with $p_i = D\, a_i \in \mathbb{Z}$. This is exact because the persistence diagram depends only on the order of the filtration values, not their scale (Remark 2.3). We also apply this in practice to the subsequent filtrations.

**(c) average neighbor degree.** The average neighbor degree of a vertex is the average of degrees of its neighboring vertices. This can be encoded as $\left(\sum_{j\neq i} M_{ij}d_j\right)/\max(1,d_i) = a_i$, or more preferably,

$$\textbf{Average Neighbor Degree: } \sum_{j\neq i} M_{ij}d_j = a_i \max(1,d_i) \qquad \forall\, i \in [n].$$

**(d) closeness centrality.** For a **connected graph**, the closeness centrality of a vertex $i$ is $n-1$ divided by the sum of shortest-path distances $\delta(i,j)$ from every $j$ to $i$. We can encode this as

$$\textbf{Closeness Centrality}: \; n-1 = a_i \sum_{j\neq i} \delta_{ij} \qquad \forall\, i \in [n].$$

Here $\delta_{ij}$ are integer variables forced to be $\delta(i,j)$ by imposing the constraints: for each source $i$, (i) $\delta_{ii} = 0$; (ii) $\delta_{ij} \geq 1$ for $j \neq i$; (iii) $|\delta_{ij} - \delta_{ik}| \leq 1$ when $M_{jk} = 1$; and (iv) every $j \neq i$ has a neighbor $p_{ij}$ with $\delta_{ip_{ij}} = \delta_{ij} - 1$. This forces $\delta_{ij} = \delta(i,j)$.

**(e) random walk centralities.** The same technique of pinning a value as the unique solution of a system covers both **Katz centrality** and **PageRank**. For a given $\alpha$, we can encode them as

$$\textbf{Katz centrality}: \; a_i = 1+\alpha \sum_{j\neq i} M_{ij}a_j, \; \forall\, i \in [n]\,; \textbf{PageRank}: \; a_i = \tfrac{1-\alpha}{n} + \alpha \sum_{j\neq i} \tfrac{M_{ij}}{d_j} a_j, \; \forall\, i \in [n].$$

In Appendix J, we verify soundness and completeness of all filtrations above on small graphs.

# 5 Experiments

**Datasets.** We evaluate on four standard graph generation benchmarks: COMM-20 [63], PLANAR [32], SBM [32], and ENZYMES [34], spanning a range of sizes, densities, and topological regimes. We adopt the SPECTRE 64:16:20 train/validation/test split convention [32] uniformly across all four datasets. Statistics and splits are deferred to Appendix H.1, see also Remark H.1 for a technical detail on implementing Algorithm 3.

## 5.1 Comparing Local and Global Generation

In this first experiment we evaluate our two PD-exact generators against each other. For each test graph, we feed its degree-based PD $(\mathcal{P}_0, \mathcal{P}_1)$ as the target to both methods, and report the success rate, wall-clock cost, and (for the local sampler) attempts before first success.

**Setup.** The local degree-filtration generator (Algorithm 3) is allowed up to 20,000 rejection attempts per target before a run is marked as failed; the global CP solver (Sec. 4) is given a 1,800 s per-target time budget. Both run on a 120-CPU-core machine with all per-target generations parallelised across workers. We report success rate over the test split, mean wall-clock per target (charging the full budget on failed runs), and, for the local sampler, mean attempts to first success on successful runs. Results under random and vertex filtrations, together with median statistics, are deferred to Appendix H.4.

Table 2: Local degree-filtration sampler vs. the global CP solver. Wall-clock means charge the full attempt or time budget on failed runs.

| | | COMM-20 | PLANAR | SBM | ENZYMES |
|---|---|---|---|---|---|
| LOCAL (Alg. 3) | Success rate (%) ↑ | 100.0 | 100.0 | 15.0 | 99.1 |
| | Wall-clock per target (s) ↓ | 0.17 | 5.88 | 1465.0 | 6.94 |
| | Attempts to first success ↓ | 49.6 | 137.5 | 4583.5 | 431.5 |
| GLOBAL (CP, Sec. 4) | Success rate (%) ↑ | 100.0 | 100.0 | **82.5** | 100.0 |
| | Wall-clock per target (s) ↓ | 0.30 | 16.35 | **654.8** | 6.85 |

**Discussion.** Table 2 shows that on COMM-20, PLANAR, and ENZYMES both methods succeed essentially everywhere, with the local sampler 1.7–2.8× faster on the smaller datasets and on par on ENZYMES. However, in SBM, the local sampler succeeds on only 15% of targets and exhausts the budget on the rest, while CP returns a valid graph on 82.5% at less than half the wall-clock. Consistent with Theorem 3.10, local moves fail when the PD encodes a tightly constrained global structure, which CP's global view recovers. The two methods are thus complementary. Appendix I.3 gives a practitioner's guide to choosing a priori among them and approximate neural generators.

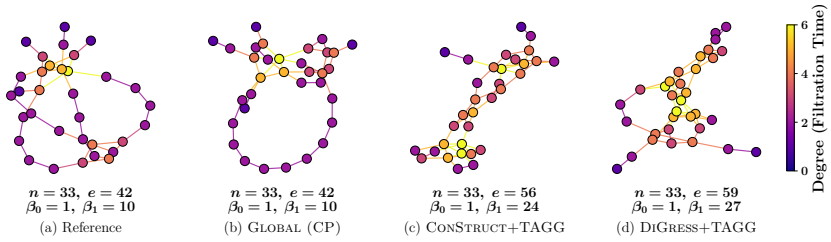



Figure 3: Qualitative example on an ENZYMES test graph. Node colour encodes degree. CP matches $(n, |E|, \beta_0, \beta_1)$ exactly, while neural variants add excess cycles. More in Appendix H.6.

### 5.2 Comparison with Neural Generative Baselines

We now compare against neural diffusion baselines that approximate PD-conditioning.

**Setup.** We compare two backbones, DIGRESS [52] and CONSTRUCT [30], both augmented with TAGG's [40] topology-aware attention module (TAM) and 1-Wasserstein PDM loss, yielding DIGRESS+TAGG and CONSTRUCT+TAGG. Both variants are trained per-dataset with PD conditioning supplied via TAM and regularised by PDM. At inference we feed each test graph's own persistence-landscape vector $\mu_{G_{\text{test}}}$ rather than the dataset average $\mu_{G'}$ used by the original TAGG, granting the neural variants the strongest possible per-target signal and a paired one-to-one correspondence with the reference. CP receives the same target PD as direct input. Neural variants are evaluated over 5 random seeds; CP is deterministic. We measure topological fidelity by the mean paired 1-Wasserstein distance under the degree filtration, $\bar{W}_1 = \frac{1}{N}\sum_i [W_1(\mathcal{P}_0(G_{\text{ref}}^{(i)}), \mathcal{P}_0(H^{(i)})) + W_1(\mathcal{P}_1(G_{\text{ref}}^{(i)}), \mathcal{P}_1(H^{(i)}))]$, with diagonal padding (the metric used as the PDM training loss). Details are deferred to Appendix H.5.

**Discussion.** Table 3 shows that CP achieves $\bar{W}_1 = 0$ on every dataset by construction, while the neural variants do not. (full table with std. dev. in Appendix H.5, Table 6) Figure 3 makes the same point per-graph: the neural variants merge two components into one and introduce excess cycles, while CP returns a different graph from the reference (many graphs share the same PD; see Appendix K) but matches every degree-PD invariant exactly. A second qualitative example, with a full per-graph breakdown of edge and cycle drift, is in Appendix H.6. In terms of runtime cost, neural variants are 3–38× faster than CP at inference on larger datasets but require 9–62 hours of per-dataset training; CP needs none. The two answer different questions: CP when exact PD-correctness is required and the constraint problem is tractable; neural variants when approximate correctness suffices and the training cost amortizes over many generations.

Table 3: GLOBAL (CP) vs. TAGG-augmented neural baselines on degree-PD $\bar{W}_1$ (0 = exact match).

| | | COMM-20 | PLANAR | SBM | ENZ. |
|---|---|---|---|---|---|
| CP | $\bar{W}_1 \downarrow$ | **0.00** | **0.00** | **0.00** | **0.00** |
| | Time (s) $\downarrow$ | 0.30 | 16.35 | 654.8 | 6.85 |
| | Train (h) $\downarrow$ | 0 | 0 | 0 | 0 |
| CONSTR. +TAGG | $\bar{W}_1 \downarrow$ | 9.35 | 43.23 | 434.48 | 17.35 |
| | Time (s) $\downarrow$ | 0.88 | 4.73 | 17.11 | 4.92 |
| | Train (h) $\downarrow$ | 9.4 | 24.3 | 59.8 | 61.6 |
| DIGRESS +TAGG | $\bar{W}_1 \downarrow$ | 4.52 | 38.77 | 464.99 | 17.06 |
| | Time (s) $\downarrow$ | 0.90 | 5.30 | 23.16 | 5.94 |
| | Train (h) $\downarrow$ | 8.9 | 32.5 | 47.6 | 49.2 |

## 6 Conclusion

We initiated a principled study of strict PD-compliant graph generation: producing graphs whose persistence diagram under a given filtration matches a target exactly. Focusing on the degree filtration, we developed two complementary generators: a local rejection sampler with soundness and completeness guarantees, and a global constraint-programming formulation with a minimal-sufficient encoding. Furthermore, we extended the global formulation to any vertex-based permutation equivariant filtration scheme. We showed that the local view, while cheap on easy instances, is fundamentally limited (Theorem 3.10) when the PD encodes tight global structure. Empirically, both methods produce graphs realizing the target PD exactly on standard benchmarks, while neural diffusion baselines with PD-conditioning regularizers leave a structural gap that more training does not close. We see this work as a first step: natural extensions include richer filtration schemes, controlled relaxations of strict compliance (e.g. bottleneck-ball targets), and using the constraint encoding as a building block inside learned generators.

## Acknowledgments and Disclosure of Funding

This project was conceived during discussions at NeurIPS 2025, with early ideas dating back to when MJ and IR were participating during 2025 AScI international summer research programme at Aalto University. MJ is partially supported by the National Science Foundation Graduate Research Fellowship (DGE-2236662). IR and VG acknowledge support from Saab-WASP (grant 411025). VG also acknowledges the Jane and Aatos Erkko foundation (grant 7001703) for their support. We are grateful to the anonymous program chair, area chairs and the reviewers for their constructive feedback and service. MJ would especially like to thank Thomas Brazelton who inspired MJ to think about the exact topological compliance for graphs. MJ would also like to thank Erin Chambers for helpful conversations as well as Yuqin Kewang, Guangyu Meng, and Shankha Mukherjee for support during rebuttal.

## References

[1] R. Anirudh, V. Venkataraman, K. N. Ramamurthy, and P. Turaga. A Riemannian Framework for Statistical Analysis of Topological Persistence Diagrams . In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1023–1031, Los Alamitos, CA, USA, July 2016. IEEE Computer Society.

[2] R. Ballester and B. Rieck. On the expressivity of persistent homology in graph learning. In G. Wolf and S. Krishnaswamy, editors, *Proceedings of the Third Learning on Graphs Conference*, volume 269 of *Proceedings of Machine Learning Research*, pages 42:1–42:31. PMLR, 26–29 Nov 2025.

[3] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Reconstructing embedded graphs from persistence diagrams. *Computational Geometry*, 90:101658, 2020.

[4] G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[5] M. Carriere, F. Chazal, Y. Ike, T. Lacombe, M. Royer, and Y. Umeda. Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2786–2796. PMLR, 26–28 Aug 2020.

[6] F. Chazal, D. Cohen-Steiner, and Q. Mérigot. Geometric inference for probability measures. *Foundations of Computational Mathematics*, 11(6):733–751, Dec 2011.

[7] F. Chazal and S. Y. Oudot. Towards persistence-based reconstruction in Euclidean spaces. In *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry*, SoCG '08, pages 232–241. ACM, 2008.

[8] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, Dec 2006.

[9] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. *Transactions of the American Mathematical Society, Series B*, 9:1006–1043, 2022.

[10] N. De Cao and T. Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[11] V. De Silva and R. Ghrist. Coverage in sensor networks via persistent homology. *Algebraic & Geometric Topology*, 7(1):339–358, 2007.

[12] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[13] T. K. Dey and Y. Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022.

[14] H. Edelsbrunner and D. Morozov. Persistent homology: Theory and practice. In *Proceedings of the European Congress of Mathematics*, 2012.

[15] I. P. Gent and B. Smith. *Symmetry breaking during search in constraint programming*. University of Leeds, School of Computer Studies Leeds, 1999.

[16] R. Ghrist, R. Levanger, and H. Mai. Persistent homology and euler integral transforms. *Journal of Applied and Computational Topology*, 2(1):55–60, Oct 2018.

[17] O. Hacquard, G. Blanchard, and C. Levrard. Statistical learning on measures: An application to persistence diagrams. *Electronic Journal of Statistics*, 20(1):49–82, 2026.

[18] A. Hatcher. *Algebraic Topology*. New York : Cambridge University Press, 2002.

[19] C. Hofer, F. Graf, B. Rieck, M. Niethammer, and R. Kwitt. Graph Filtration Learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4314–4323. PMLR, 13–18 Jul 2020.

[20] C. Hofer, R. Kwitt, M. Niethammer, and A. Uhl. Deep learning with topological signatures. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1633–1643, Red Hook, NY, USA, 2017. Curran Associates Inc.

[21] M. Horn, E. De Brouwer, M. Moor, Y. Moreau, B. Rieck, and K. Borgwardt. Topological graph neural networks. *arXiv preprint arXiv:2102.07835*, 2021.

[22] J. Immonen, A. Souza, and V. Garg. Going beyond persistent homology using persistent homology. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 63150–63173. Curran Associates, Inc., 2023.

[23] M. Ji, I. Roy, and V. K. Garg. Contraction and hourglass persistence for learning on graphs, simplices, and cells. In *The Fourteenth International Conference on Learning Representations*, 2026.

[24] M. Ji, A. H. Souza, and V. K. Garg. Graph persistence goes spectral. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[25] M. Ji, A. H. Souza, and V. K. Garg. On topological descriptors for graph products. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[26] W. Jin, R. Barzilay, and T. Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International conference on machine learning*, pages 4839–4848. PMLR, 2020.

[27] J. Jo, S. Lee, and S. J. Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International conference on machine learning*, pages 10362–10383. PMLR, 2022.

[28] B. Li, D. Ofori-Boateng, Y. R. Gel, and J. Zhang. A hybrid approach for transmission grid resilience assessment using reliability metrics and power system local network topology. *Sustainable and Resilient Infrastructure*, 6(1-2):26–41, 2021.

[29] B. Liang, S. Chen, L. Gui, H. Wang, Y. Yu, R. Xu, and K.-F. Wong. Centrality-guided pre-training for graph. In *The Thirteenth International Conference on Learning Representations*, 2025.

[30] M. Madeira, C. Vignac, D. Thanou, and P. Frossard. Generative modelling of structurally constrained graphs. *Advances in Neural Information Processing Systems*, 37:137218–137262, 2024.

[31] J. P. Marques-Silva and K. A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on computers*, 48(5):506–521, 1999.

[32] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. In *International Conference on Machine Learning*, pages 15159–15179. PMLR, 2022.

[33] G. Meng, P. Gu, P. Liang, J. P. Lalor, E. W. Chambers, and D. Z. Chen. TopoCL: Topological Contrastive Learning for Medical Imaging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 42681–42690, June 2026.

[34] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[35] F. Mémoli and T. Needham. Distance distributions and inverse problems for metric measure spaces. *Studies in Applied Mathematics*, 149(4):943–1001, 2022.

[36] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon. Permutation invariant graph generation via score-based generative modeling. In *International conference on artificial intelligence and statistics*, pages 4474–4484. PMLR, 2020.

[37] T. Ophelders and A. Schenfisch. Sweeping orders for simplicial complex reconstruction, 2025.

[38] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6(1), Aug. 2017.

[39] T. Papamarkou, T. Birdal, M. M. Bronstein, G. E. Carlsson, J. Curry, Y. Gao, M. Hajij, R. Kwitt, P. Lio, P. Di Lorenzo, V. Maroulas, N. Miolane, F. Nasrin, K. Natesan Ramamurthy, B. Rieck, S. Scardapane, M. T. Schaub, P. Veličković, B. Wang, Y. Wang, G. Wei, and G. Zamzmi. Position: Topological deep learning is the new frontier for relational learning. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 39529–39555. PMLR, 21–27 Jul 2024.

[40] J. Park, D. Lee, Y. Song, G. Wu, and W. H. Kim. Topology-aware graph diffusion model with persistent homology. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[41] J.-F. Puget. Symmetry breaking revisited. In *International Conference on Principles and Practice of Constraint Programming*, pages 446–461. Springer, 2002.

[42] J.-C. Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, volume 94, pages 362–367, 1994.

[43] T. J. Ritchie and S. J. Macdonald. The impact of aromatic ring count on compound developability–are too many aromatic rings a liability in drug design? *Drug discovery today*, 14(21-22):1011–1020, 2009.

[44] F. Russold and M. Kerber. Graphcode: Learning from multiparameter persistent homology using graph neural networks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[45] A. Schenfisch, B. T. Fasy, S. Micka, D. L. Millman, and L. Williams. Efficient graph reconstruction and representation using augmented persistence diagrams. In Y. Bahoo and K. Georgiou, editors, *Proceedings of the 34th Canadian Conference on Computational Geometry, CCCG 2022, Toronto Metropolitan University, Toronto, Ontario, Canada, August 25-27, 2022*, pages 284–292, 2022.

[46] M. Simonovsky and N. Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

[47] A. E. Sizemore, C. Giusti, A. Kahn, J. M. Vettel, R. F. Betzel, and D. S. Bassett. Cliques and cavities in the human connectome. *Journal of computational neuroscience*, 44(1):115–145, 2018.

[48] J. Southern, J. Wayland, M. Bronstein, and B. Rieck. Curvature filtrations for graph generative model evaluation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[49] J. Townsend, C. P. Micucci, J. H. Hymel, V. Maroulas, and K. D. Vogiatzis. Representation of molecular structures with persistent homology for machine learning applications in chemistry. *Nature communications*, 11(1):3230, 2020.

[50] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 2014.

[51] Y. Verma, A. H. Souza, and V. Garg. Topological neural networks go persistent, equivariant and continuous. In *ICML 2024 AI for Science Workshop*, 2024.

[52] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

[53] G. Villines. Topological signatures of structural risk: Persistent homology detects regime-dependent patterns in trade networks and catastrophe loss fields. *Available at SSRN 6299678*, 2026.

[54] A. Z. Wagner. Constructions in combinatorics via neural networks. *arXiv preprint arXiv:2104.14516*, 2021.

[55] B. Wang, S. Zheng, J. Wu, J. Li, and F. Pan. Inverse design of catalytic active sites via interpretable topology-based deep generative models. *npj Computational Materials*, 11(1):147, 2025.

[56] R. Wang, J. Chen, K. Gao, Y. Hozumi, C. Yin, and G.-W. Wei. Analysis of sars-cov-2 mutations in the united states suggests presence of four substrains and novel variants. *Communications biology*, 4(1):228, 2021.

[57] G.-W. Wei. Topological AI forecasting of future dominating viral variants. *ArXiv*, pages arXiv–2209, 2022.

[58] P. Woldemariam and N. Attoh-Okine. Topological data analysis for railway track geometry safety and maintenance. *Journal of Computing in Civil Engineering*, 39(5):04025073, 2025.

[59] K. Xia, X. Feng, Y. Tong, and G. W. Wei. Persistent homology for the quantitative prediction of fullerene stability. *Journal of computational chemistry*, 36(6):408–422, 2015.

[60] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[61] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

[62] Z. Yan, T. Ma, L. Gao, Z. Tang, Y. Wang, and C. Chen. Neural approximation of graph topological features. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[63] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

[64] Q. Zhao, Z. Ye, C. Chen, and Y. Wang. Persistence enhanced graph neural network. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2896–2906. PMLR, 26–28 Aug 2020.

# Exact Topological Compliance: Generating Persistence-Equivalent Graphs (Appendix)

## A Limitations and Future Directions

There are interesting graph properties that are not preserved under PD-equivalent graphs. For example, planarity and information about clustering are not necessarily invariants under PD-equivalence. The generation methods discussed in this paper are not guaranteed to maintain them either. As a future direction, we could aim to generate graphs that preserves both topological properties and other non-topological properties. In Appendix K, we tested how diverse graphs in a single degree-based PD-equivalence class can be. Indeed, within one PD-equivalence class, clustering, triangle counts, spectra, and planarity all vary.

There are also a number of future directions that would be interesting to explore, for which our work did not focus on.

**Higher-order PDs.** We restricted to filtrations on graphs, where the PDs of dimension $\geq 2$ are empty. One may be interested to extend this to filtrations on simplicial complexes. As there are higher simplices, tracking their effects would require matrix-reduction procedures, which increases both computational cost and the chance of infeasible states. To handle higher-order PDs on simplicial complexes, we would need to incorporate additional constraints in Section 4. We anticipate it would be challenging to characterize simple combinatorial constraints for the higher-order PDs, and the CP solver would certainly take longer to solve.

**Scalability.** Exact global generation can be computationally expensive, and the CP formulation is not expected to scale uniformly to large or difficult instances. In Appendix I, we noted that the encoding size is polynomial; solving is NP-hard in general (it subsumes SAT), so no closed-form runtime bound applies. In regimes where both exact methods fail (harder SBM instances are the concrete example), neural sampling is the only tractable choice, at the cost of exact PD compliance.

**General filtrations.** The soundness and completeness of the encoding in Theorem 4.2 is independent of the given $\Phi$. However, solve cost depends on the encoding complexity: degree is linear, whereas triangle and clustering add $O(n^3)$ auxiliaries and are costlier (see Appendix J). One could imagine the costs would vary across general filtrations.

**Neural interface.** Exact persistence-diagram agreement does not imply that the generated graphs preserve other relevant properties. Coupling our methods with a neural generator to preserve properties topological summaries miss, such as planarity and spectral data, is a promising direction for future work, as is a downstream end-to-end demonstration. Since one motivation for exact compliance was to find counter-examples, one may ask if neural interface (along the themes of [54]) can help to find such examples.

## B Broader Impact Statement

This paper presents work whose goal is to advance the problem of strictly Persistence-Diagram compliant graph generations. As highlighted in the introduction, topological descriptors have a wide variety of usage in biology, infrastructure networks, and more. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

## C Declaration of LLM Usage

We used large language models (LLMs) for help in the coding and visualization part of the paper. We also consulted LLMs in the writing of the paper, where we double checked the written statements. The proofs in Appendix D and Appendix F were written by human.

# D Proofs in Section 3

## D.1 Proofs in Section 3.1

*Proof of Theorem 3.1.* For the first statement, suppose the algorithm terminates at the end, then we have created a sequence of subgraphs at each stage as:

$$H_\bullet : H_{-1} = \emptyset \subseteq H_0 \subseteq ... \subseteq H.$$

Here we verify that $\mathcal{P}(H_\bullet) = (\mathcal{P}_0, \mathcal{P}_1)$. Indeed, at each stage $k$, each vertex addition step adds in un-filled tuples of the form $(k, -)$ for each $(k, d) \in \mathcal{P}_0$. For each $(b, k) \in \mathcal{P}_0$ processed at the component merging step, the edge added in the component merging step goes from a component born at time $b$ to one born at time $\leq b$, which has the effect of killing off the prior component. If they both happened to have the same birth time, we can without loss still kill off the first component. Thus, we see that the component merging step perfectly aligns with the $(b, k)$'s processed in $\mathcal{P}_0$. The same discussion applies to the cycles $(k, \infty)$ processed in the cycle addition step.

Thus, we see that as long as the algorithm terminates, then by design the result filtration $H_\bullet$ has persistence pairs strictly equal to $(\mathcal{P}_0, \mathcal{P}_1)$.

For the second statement, by induction the proof reduces to the following case - suppose we have produced a sequence of subgraphs from the algorithm

$$(G'_k)_\bullet : \emptyset = G'_{-1} \subseteq G'_0 \subseteq ... \subseteq G'_k, \quad \mathcal{P}^i((G'_k)_\bullet) = \mathcal{P}^i(G_\bullet) \text{ for } -1 \leq i \leq k,$$

Suppose we have $G'_{k+1} \supseteq G'_k$ such that $\mathcal{P}^{k+1}((G'_{k+1})_\bullet) = \mathcal{P}^{k+1}(G_\bullet)$, we seek to show that $G'_k$ can be extended to $G'_{k+1}$ **using the moves described in the algorithm**. Note here the base case $k = -1$ is vacuously true.

Indeed, let $H$ be $G'_{k+1} - G'_k$ (which could have open edges missing a vertex) and suppose $H$ has $m$ edges.

Although in the step $G'_k \subseteq G'_{k+1}$, the entire $H$ is added at once, we can algorithmically think of this as adding individual vertices and edges one at a time. The theory of persistent homology ensures that the final persistence pairs produced at the end are well-defined independent of the order of simplices added, as long as whenever an edge $e = (u, v)$ is added, the vertices $u$ and $v$ are added first. Thus we can think of the step $G'_k \subseteq G'_{k+1}$ as

$$G'_k \subseteq G'_{k,0} \subseteq G'_{k,1} \subseteq ... \subseteq G'_{k,m} = G'_{k+1}$$

where $G'_k \subseteq G'_{k,0}$ adds all the vertices of $H$ in, and for $i \geq 0$, $G'_{k,i} \subseteq G'_{k,i+1}$ adds in exactly one edge from $H$. The step $G'_k \subseteq G'_{k,0}$ is exactly the first two steps of the algorithm.

Now from the step $G'_{k,0} \to G'_{k,1}$, an edge either kills a tuple in $\mathcal{P}_0$ or makes a tuple in $\mathcal{P}_1$. If the edge $e = (v, w)$ adds a cycle, then $v, w$ belongs to the same component at $G'_{k,0}$. Observe that in general if two vertices $v, w$ are in the same component of graph $A$, and $A \subseteq B$, then $v, w$ are in the same component of $B$. Thus, it is possible that an edge created in the **cycle creation** step of the algorithm links $v$ and $w$. If the edge fills a tuple $(b, -)$ to $(b, k+1)$, then it is drawn from an undead component containing $v$ whose birth time is at $b$ to a component containing $w$ that is born earlier. Now observe that in general if the component of $v \in A$ has birth time $b$, and $A \subseteq B$, then $v \in B$ also has birth time $\leq b$ (i.e. the birth time can only decrease). Thus, this can happen in the first tuple encountered in the **componet merging** step.

We can inductively align each step $G'_{k,i} \subset G'_{k,i+1}$ such that the edge $e$ added, if contributing to the death time of a component, is the $j$-th step in the for loop of the **component merging** step, and $e$ is the $j$-th component-killing-edge added. Indeed, by the second observation in the last paragraph, it suffices to note that the cycle-creating-edges does not link different components. Thus, we conclude the order of component-killing-edges added in $G'_{k,0} \to G'_{k,1}$ can be realized as the algorithmic steps occurring in the **component merging** step. As argued, above, the cycle-creating edges can all be deferred to the **cycle creating** step and are hence accounted as well. This concludes the proof. □

*Proof of Proposition 3.3.* Suppose by induction we have iteratively built a graph $H_{k-1}$. If we just view $H_{k-1}$ as the final graph, then by the first part of Theorem 3.1, we know that $\mathcal{P}(H_{-1} \subseteq H_0 \subseteq ...H_{k-1}) = \mathcal{P}^k(G_\bullet)$ (†).

Now, clearly the vertex addition step can always be carried out, as we are just adding disjoint vertices to the graph. The **component merging** step can always be carried out. Indeed, by (†), we know the components in $H_{k-1}$ and the components in $G_{k-1}$ have the same multi-set of birth times. Thus, we can always find the desired edge between two components in the component merging step.

Finally, for the **cycle addition** step: We first note that, in general, for non-empty connected graphs $A$ and $B$ where $A$ has more than 1 vertex, let $C$ denote linking $A$ and $B$ by an edge. Observe that $C$ always has room for cycles since we can draw an edge between any vertex in $A$ not linked to $B$ to vertices in $B$.

Now we observe that for a graph $A$ with isolated vertices, write $A = A' \sqcup A''$ where $A'$ has no isolated vertices and $A''$ is a discrete graph (no edges). Then clearly $\mathrm{cap}^{\mathrm{cyc}}(A) = \mathrm{cap}^{\mathrm{cyc}}(A')$ as isolated vertices cannot support cycles.

Now let $H'_k \supseteq H_{k-1}$ be the new graph extending $H_{k-1}$ that is created after the **component merging** step. Observe that all the edges $e$ added in the **component merging step** fall into one of the following two scenarios (1) $e$ is drawn between two isolated vertices or (2) $e$ is drawn between two connected graphs $A$ and $B$ where one of them has more than 1 vertex. Regardless, we see that the cycle capacity of the graph can only increase, and $\mathrm{cap}^{\mathrm{cyc}}(H'_k) \geq \mathrm{cap}^{\mathrm{cyc}}(H_k)$. Finally, the **cycle addition** step can succeed if and only if $\mathrm{cap}^{\mathrm{cyc}}(H'_k)$ is $\geq$ the number of tuples of the form $(k, \infty)$ in $\mathcal{P}_1$. This holds since it holds for $\mathrm{cap}^{\mathrm{cyc}}(H_k)$. □

*Proof of Corollary 3.4.* By Proposition 3.3, we know that we can successfully build $H_{k+1}$ from a given $H_k$ if $\mathrm{cap}^{\mathrm{cyc}}(H_k)$ is greater than or equal to the number of tuples of the form $(k+1, \infty)$ in $\mathcal{P}_1$. If $\mathcal{P}_1(G_\bullet)$ is the empty list, this number is in particular zero. Since the cycle capacity of a graph is non-negative, it is above greater than or equal to 0. Thus, we can always successfully build $H_{k+1}$ from a given $H_k$ here. The statement then follows from induction. □

### D.2 Proofs in Section 3.2

*Proof of Proposition 3.6.* It suffices to show that any non-trivial vertex death at a time must also be accompanied with trivial deaths. We suppose for contradiction that this statement is false.

Let $G_\bullet$ be a vertex-based filtration of such counter-example at stage $G_k \subseteq G_{k+1}$. Now suppose the non-trivial death occurs by an edge $e = (v, w)$ in the steps building $G_k$ to $G_{k+1}$. Without loss, we can also take $v \in V(G_k)$ and $w \in V(G_{k+1} - G_k)$, because $f$ is a vertex-based filtration. If $w$ has not been linked to any edge yet, this can only constitute a trivial death as $w$ is born later than $v$. Thus, it must be the case that $w$ had an edge connected to it already. In the first time $w$ is connected to some other vertex $v_0$, either $w$ dies because $v_0$ is younger (giving a trivial pair) or $w$ and $v_0$ are born at the same time (still giving a trivial pair). Thus, we see we would get a trivial pair regardless. □

*Proof of Theorem 3.7.* By design, Algorithm 2 realizes a subset of possible steps taken in Algorithm 1. Thus Theorem 3.1 implies that the final $H_\bullet$, if created, has $\mathcal{P}(H_\bullet) = (\mathcal{P}_0, \mathcal{P}_1)$. Now we wish to ensure that $H_\bullet$ is a vertex-based filtration. Indeed, in the **component merging** step and the **cycle creation** step, every edge that is added is conditioned have at least one of its vertices being supported in the newly created vertices at that stage. Thus, the filtration is vertex-based.

For any vertex-based filtration $H'_\bullet$ that is PD-equivalent to $G_\bullet$, we now argue how Algorithm 2 can create this. Indeed, if the **minimum vertex pivot** step is not imposed, then clearly Theorem 3.1 implies that the algorithm can recreate $H'_\bullet$, as we had only imposed the minimum constraints to make them vertex-based. Thus, it suffices to show that this minimum vertex pivot can always be realized as the first step in $H'_k \subseteq H'_{k+1}$ for any vertex-based filtration (provided a vertex death has happened).

Indeed if vertex deaths occurred, then the component merging step has to come with the addition of new edges $\{e_1, ..., e_r\}$. Now look at all components that the vertices of $e_i$'s belong in for all $i$ and find the one with smallest birth time. Since the list of edges is non-empty, we can find a possibly non-unique component.

Now we claim we can perform the minimum vertex pivot there. Indeed, let $B$ be the birth time of this component. By construction we know that $B \leq b_{min}$, as $b_{min}$ takes the minimum over a subset of the edges. If $B < b_{min}$, then this can always be realize in the minimum vertex generation step of the algorithm. If $B = b_{min}$, we need to check that not all components born at $b_{min}$ are set to die at $k + 1$. Indeed, suppose for contradiction they are, then to kill off the last one, you need to make an edge from a component born earlier than $b_{min}$, which implies $B < b_{min}$. Thus, we see the step for minimum vertex pivot is always possible and can go first.

Finally for the last sentence: clearly we can always do vertex addition, and the ability to do minimum vertex pivot follows from Proposition 3.6. The ability to succeed in component merging follows similarly as in Proposition 3.3 from induction and the fact that the newly created vertices from stage $k-1$ to stage $k$ are the same in the hidden filtration and the filtration we are creating. □

*Proof of Corollary 3.8.* This again follows similarly as to the proof of Corollary 3.8. Since we do not need to add cycles if $\mathcal{P}_1$ is empty, Theorem 3.7 implies the algorithm always terminates. □

### D.3 Proofs in Section 3.3

*Proof of Theorem 3.9.* By design, Algorithm 3 realizes a subset of possible steps taken in Algorithm 2. Thus Theorem 3.7 implies that the final $H_\bullet$, if created, is vertex-based and has $\mathcal{P}(H_\bullet) = (\mathcal{P}_0, \mathcal{P}_1)$. This in particular implies $H$ and $G$ have the same number of vertices born at each stage, and the same number of edges added at each stage (†).

For the first statement, it suffices to show that the filtration is degree-based. Since this is already vertex based, it suffices to show that the degree of each vertex added at stage $k$ is $k$. This is exactly what the capacity assigned to each vertex means. If the algorithm can terminate at the end, then the capacity of each node has to be 0 by (†) and pigeonhole's principle. Thus, a node born at time $k$ will obtain $k$ many edges incident to it in the final graph. This gives the desired degree-based filtration.

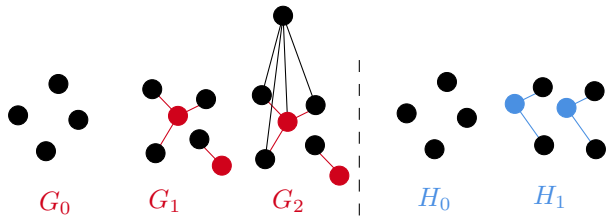
For the second sentence: A degree-based filtration is in particular a vertex-based filtration. By Theorem 3.7, it can be realized by Algorithm 2. On the other hand, any realization of this in Algorithm 2 automatically passes the capacity check, as the original filtration is a degree-based filtration. Thus, it can be realized by Algorithm 3. □

### D.4 Proofs in Section 3.4

*Proof of Theorem 3.10.* We split the proof into the following cases.

**For General Filtrations:** Suppose for contradiction that such Algorithm A exists. Consider $G_\bullet = (G_{-1} \subseteq G_0 \subseteq G_1 \subseteq G_2)$ as given in Figure 2. By (1) and (2), we have that $(H_{-1} \subseteq H_0 \subseteq H_1) \in \mathcal{O}_1(G_\bullet)$. By (1), we know that there must exist some $H_2$ extending $H_1$ such that $\mathcal{P}(H_{-1} \subseteq H_0 \subseteq H_1 \subseteq H_2) = \mathcal{P}(G_\bullet)$. However, as indicated by the Example 3.2, no such example can exist.

**For Vertex-Based Filtrations**: Suppose for contradiction that such Algorithm A exists. Consider $G_\bullet$ in the following diagram:



We have that $\mathcal{P}(G_{-1} \subseteq G_0 \subseteq G_1) = \mathcal{P}(H_{-1} \subseteq H_0 \subseteq H_1)$. Thus by (1) and (2), there should exist $H_2$ extending $H_{-1} \subseteq H_0 \subseteq H_1$ such that $\mathcal{P}(H_{-1} \subseteq H_0 \subseteq H_1 \subseteq H_2) = \mathcal{P}(G_\bullet)$. We see from the persistence pair that this requires the addition of 4 edges (one can count the number of edges added based on how many vertex deaths and cycle births happened). We see from $\mathcal{P}(G_\bullet)$ that 3 cycles are made and 1 trivial vertex death occurred in going from $G_1 \subseteq G_2$. However, by pigeonhole's principle, there is no way to adjoin a new vertex $v$ to $H_1$ along with 4 edges from $H_1$ to $v$ such that this happens.

**For Degree-Based Filtrations:** Suppose for contradiction that such Algorithm A exists. Observe that (1) and (2) implies a sub-filtration of a degree-based filtration should be a degree-based filtration. However, this is not true. For example, consider the degree based filtration of either Graph 1 or Graph 2 in Figure 1 and truncate up to $t = 2$. There are degree 1 vertices that are spawned at $t = 2$, which is a contradiction. □

### D.5 Additional Remark for Section 3

**Remark D.1.** Instead of asking Algorithm 1 to generate PD-equivalent graph filtrations, we could ask Algorithm 1 to defer the cycle addition to a later stage if it ran out of cycles to add. This would typically make a graph filtration that is no longer PD-equivalent, but we would expect it to be a closer approximation of the original graph filtration than many other models in the literature.

## E Pseudocode of the Local Generators

Here, we provide the full pseudocode for the three local generators of Section 3. Algorithm 1 is the general-filtration generator. In Algorithm 2, the changes with respect to Algorithm 1 are shown in violet. Later, in Algorithm 3, the further changes with respect to Algorithm 2 are shown in teal.

**Algorithm 1: General filtration** generator. PD-equivalent under any general filtration.

**Input:** $(\mathcal{P}_0, \mathcal{P}_1)$ persistence pairs from a hidden filtration
**Output:** $H, f_V, f_E$ specifying the filtration of the graph $H$, or FAIL
**Pre-process:** $B_0(k) = \{(k,d) \in \mathcal{P}_0\}$, $D_0(k) = \{(b,k) \in \mathcal{P}_0\}$, $B_1(k) = |\{(k,\infty) \in \mathcal{P}_1\}|$;
**Initialize:** $H \leftarrow \emptyset$; union-find $\mathcal{U}$; per-node arrays $\mathbf{e}$ (exists), $\mathbf{c}$ (comp-id), $\mathbf{b}$ (comp-birth); adjacency $A$;
**for** $k = 0, \ldots, n$ **(filtration steps)**
  **foreach** $(k,d) \in B_0(k)$ **(Vertex Add.)**
    add $v$; $f_V(v) \leftarrow k$; $\mathbf{e}[v] \leftarrow 1$; $\mathbf{c}[v] \leftarrow v$; $\mathbf{b}[v] \leftarrow k$;
  **foreach** $(b,k) \in$ *shuffle*$(D_0(k))$ **(Comp. Merge)**
    $I_{\mathrm{m}}[i,j] \leftarrow \mathbf{e}[i] \wedge \mathbf{e}[j] \wedge (\mathbf{b}[i] = b) \wedge (\mathbf{b}[j] \leq b) \wedge (\mathbf{c}[i] \neq \mathbf{c}[j]) \wedge \neg A[i,j]$;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{m}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$; $\mathcal{U}.\mathrm{union}(u,v)$;
    $\mathbf{b}[\mathcal{U}.\mathrm{find}(u)] \leftarrow \min(\mathbf{b}[u], \mathbf{b}[v])$;
  **for** $j = 1, \ldots, B_1(k)$ **(Cycle Add.)**
    $I_{\mathrm{c}}[i,j] \leftarrow \mathbf{e}[i] \wedge \mathbf{e}[j] \wedge (\mathbf{c}[i] = \mathbf{c}[j]) \wedge \neg A[i,j]$;
    **if** $I_{\mathrm{c}} = \mathbf{0}$ **then return** FAIL;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{c}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$;
**return** $H, f_V, f_E$;

**Algorithm 2: Vertex-based filtration** generator. Changes with respect to Alg. 1 are in violet.

**Input:** $(\mathcal{P}_0, \mathcal{P}_1)$ persistence pairs from a hidden vertex-based filtration
**Output:** $H, f_V, f_E$ specifying the vertex-based filtration of the graph $H$, or FAIL
**Pre-process:** $B_0(k) = \{(k,d) \in \mathcal{P}_0\}$, $D_0(k) = \{(b,k) \in \mathcal{P}_0\}$, $B_1(k) = |\{(k,\infty) \in \mathcal{P}_1\}|$;
**Initialize:** $H \leftarrow \emptyset$; union-find $\mathcal{U}$; per-node arrays $\mathbf{e}$ (exists), $\mathbf{c}$ (comp-id), $\mathbf{b}$ (comp-birth), $\mathbf{n}$ (newness); adjacency $A$;
**for** $k = 0, \ldots, n$ **(filtration steps)**
  $\mathbf{n} \leftarrow \mathbf{0}$;
  **foreach** $(k,d) \in B_0(k)$ **(Vertex Add.)**
    add $v$; $f_V(v) \leftarrow k$; $\mathbf{e}[v] \leftarrow 1$; $\mathbf{c}[v] \leftarrow v$; $\mathbf{b}[v] \leftarrow k$; $\mathbf{n}[v] \leftarrow 1$;
  **if** $D_0(k) \neq \emptyset$ **(Min. Vertex Pivot) then**
    $b_{\min} \leftarrow \min\{b : (b,k) \in D_0(k)\}$;
    $\beta = b_{\min} - 1$ if all components with birth $b_{\min}$ are dying at $k$, else $\beta = b_{\min}$;
    $I_{\mathrm{p}}[i,j] \leftarrow (\mathbf{b}[i] \leq \beta) \wedge \neg\mathbf{n}[i] \wedge \mathbf{n}[j] \wedge \neg A[i,j]$;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{p}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$; $\mathcal{U}.\mathrm{union}(u,v)$;
    $\mathbf{b}[\mathcal{U}.\mathrm{find}(u)] \leftarrow \min(\mathbf{b}[u], \mathbf{b}[v])$;
  **foreach** $(b,k) \in$ *shuffle*$(D_0(k))$ **(Comp. Merge)**
    $I_{\mathrm{m}}[i,j] \leftarrow \mathbf{e}[i] \wedge \mathbf{e}[j] \wedge (\mathbf{b}[i] = b) \wedge (\mathbf{b}[j] \leq b) \wedge (\mathbf{c}[i] \neq \mathbf{c}[j]) \wedge \neg A[i,j] \wedge (\mathbf{n}[i] \vee \mathbf{n}[j])$;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{m}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$; $\mathcal{U}.\mathrm{union}(u,v)$;
    $\mathbf{b}[\mathcal{U}.\mathrm{find}(u)] \leftarrow \min(\mathbf{b}[u], \mathbf{b}[v])$;
  **for** $j = 1, \ldots, B_1(k)$ **(Cycle Add.)**
    $I_{\mathrm{c}}[i,j] \leftarrow \mathbf{e}[i] \wedge \mathbf{e}[j] \wedge (\mathbf{c}[i] = \mathbf{c}[j]) \wedge \neg A[i,j] \wedge (\mathbf{n}[i] \vee \mathbf{n}[j])$;
    **if** $I_{\mathrm{c}} = \mathbf{0}$ **then return** FAIL;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{c}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$;
**return** $H, f_V, f_E$;

**Algorithm 3: Degree filtration** generator. Changes with respect to Alg. 1 are in violet; further changes with respect to Alg. 2 are in teal.

**Input:** $(\mathcal{P}_0, \mathcal{P}_1)$ persistence pairs from a hidden degree-based filtration
**Output:** $H, f_V, f_E$ specifying the degree-based filtration of the graph $H$, or FAIL
**Pre-process:** $B_0(k) = \{(k,d) \in \mathcal{P}_0\}$, $D_0(k) = \{(b,k) \in \mathcal{P}_0\}$, $B_1(k) = |\{(k,\infty) \in \mathcal{P}_1\}|$;
**Initialize:** $H \leftarrow \emptyset$; union-find $\mathcal{U}$; per-node arrays $\mathbf{e}$ (exists), $\mathbf{c}$ (comp-id), $\mathbf{b}$ (comp-birth), $\mathbf{n}$ (newness), $\boldsymbol{\kappa}$ (capacity); adjacency $A$;
**for** $k = 0, \ldots, n$ **(filtration steps)**
  $\mathbf{n} \leftarrow \mathbf{0}$;
  **foreach** $(k,d) \in B_0(k)$ **(Vertex Add.)**
    add $v$; $f_V(v) \leftarrow k$; $\mathbf{e}[v] \leftarrow 1$; $\mathbf{c}[v] \leftarrow v$; $\mathbf{b}[v] \leftarrow k$; $\mathbf{n}[v] \leftarrow 1$; $\boldsymbol{\kappa}[v] \leftarrow k$;
  **if** $D_0(k) \neq \emptyset$ **(Min. Vertex Pivot) then**
    $b_{\min} \leftarrow \min\{b : (b,k) \in D_0(k)\}$;
    $\beta = b_{\min} - 1$ if all components with birth $b_{\min}$ are dying at $k$, else $\beta = b_{\min}$;
    $I_{\mathrm{p}}[i,j] \leftarrow (\mathbf{b}[i] \leq \beta) \wedge \neg\mathbf{n}[i] \wedge \mathbf{n}[j] \wedge \neg A[i,j] \wedge (\boldsymbol{\kappa}[i] > 0 \wedge \boldsymbol{\kappa}[j] > 0)$;
    **if** $I_{\mathrm{p}} = \mathbf{0}$ **then return** FAIL;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{p}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$; $\mathcal{U}.\mathrm{union}(u,v)$;
    $\mathbf{b}[\mathcal{U}.\mathrm{find}(u)] \leftarrow \min(\mathbf{b}[u], \mathbf{b}[v])$; $\boldsymbol{\kappa}[u] \mathrel{-}= 1$; $\boldsymbol{\kappa}[v] \mathrel{-}= 1$;
  **foreach** $(b,k) \in$ *shuffle*$(D_0(k))$ **(Comp. Merge)**
    $I_{\mathrm{m}}[i,j] \leftarrow \mathbf{e}[i] \wedge \mathbf{e}[j] \wedge (\mathbf{b}[i] = b) \wedge (\mathbf{b}[j] \leq b) \wedge (\mathbf{c}[i] \neq \mathbf{c}[j]) \wedge \neg A[i,j]$
      $\wedge (\mathbf{n}[i] \vee \mathbf{n}[j]) \wedge (\boldsymbol{\kappa}[i] > 0 \wedge \boldsymbol{\kappa}[j] > 0)$;
    **if** $I_{\mathrm{m}} = \mathbf{0}$ **then return** FAIL;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{m}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$; $\mathcal{U}.\mathrm{union}(u,v)$;
    $\mathbf{b}[\mathcal{U}.\mathrm{find}(u)] \leftarrow \min(\mathbf{b}[u], \mathbf{b}[v])$; $\boldsymbol{\kappa}[u] \mathrel{-}= 1$; $\boldsymbol{\kappa}[v] \mathrel{-}= 1$;
  **for** $j = 1, \ldots, B_1(k)$ **(Cycle Add.)**
    $I_{\mathrm{c}}[i,j] \leftarrow \mathbf{e}[i] \wedge \mathbf{e}[j] \wedge (\mathbf{c}[i] = \mathbf{c}[j]) \wedge \neg A[i,j]$
      $\wedge (\mathbf{n}[i] \vee \mathbf{n}[j]) \wedge (\boldsymbol{\kappa}[i] > 0 \wedge \boldsymbol{\kappa}[j] > 0)$;
    **if** $I_{\mathrm{c}} = \mathbf{0}$ **then return** FAIL;
    $(u,v) \sim \mathrm{Unif}(I_{\mathrm{c}})$; $A[u,v] \leftarrow 1$; $f_E(u,v) \leftarrow k$; $\boldsymbol{\kappa}[u] \mathrel{-}= 1$; $\boldsymbol{\kappa}[v] \mathrel{-}= 1$;
**return** $H, f_V, f_E$;

## F Proof for Correctness of Encoding (Section 4)

In this section, we will prove the correctness of the encoder in Section 4 (Theorem 4.2). **We first prove the soundness of the encoder**, and we will break down the steps in a sequence of lemmas.

We first prove Lemma F.1 as a preliminary step.

**Lemma F.1.** Constraints $\mathcal{C}_1$ to $\mathcal{C}_6$ implies $\mathcal{C}_7$ **(ancestry)**: Each slot at stage $k-1$ has exactly one successor at stage $k$, and the successor inherits all its vertices.

*Proof of Lemma F.1.* Suppose for contradiction that $\mathcal{C}_7$ does not hold, then there exists two vertices $v, w$ at slot $c$ at stage $k-1$ such that $v$ and $w$ belong in two different slots $d$ and $d'$ at stage $k$ (Here we implicitly used $\mathcal{C}_2$).

By $\mathcal{C}_4$, we know there is a sequence of edges connecting $v$ to $w$ and write $v \to v_1 \to ... \to v_n \to w$ the path. Since $v$ and $w$ belong in two different component, there must exist a pair of adjacent vertices in the path here that belong in two different slots.

Thus, we without loss reduce to the case where $v < w$ share an edge, so $M_{vw} = 1$. Now by $\mathcal{C}_3$, we know that at $k+1$, we must have

$$M_{vw} = 1 \leq 1 - |x_{k+1,cv} - x_{k+1,cw}|$$

for all $c \in [\mathrm{cc}_k]$. However, since $v$ and $w$ belong in two different slots, there is a slot $d$ such that $x_{k+1,dv} = 1$ and $x_{k+1,dw} = 0$, which implies

$$M_{vw} = 1 \leq 1 - |1 - 0| = 0,$$

which is a contradiction. □

**Definition F.2.** Let $\{M_{ij}\}_{i<j}$ be the data specifying a graph $H$, whose vertex set is still $V = [n]$. We write $M(k)$ to denote the collection of $M_{ij}$'s such that $\Phi_M(i), \Phi_M(j) \leq k$. In other words, $M(k)$ specifies the induced subgraph $H_k$ of $H$ of vertices whose value is $\leq k$.

On the other hand, given the additional data $x_{kcv}$ produced by the constraints. For each $k$, we can define $H_k^{\mathrm{cp}}$ as the subgraph of $H$ whose vertices are the active vertices $V_k \subseteq V$, an edge $e = (v, w) \in H_k^{\mathrm{cp}}$ if $v, w$ belongs to the same slot at stage $k$ and $M_{vw} = 1$.

We also need the following lemma.

**Lemma F.3.** Following the notation in Definition F.2, the data of $M_{ij}$ and $x_{kcv}$ produced by constraints $\mathcal{C}_1, ..., \mathcal{C}_6$ satisfies the following:

1. The active vertices at stage $k$ are all vertices $v$ with $\Phi_M(v) = a_v \leq k$.
2. Every slot at stage $k$ has to have at least 1 vertex.
3. The number of slots at stage $k$ is the number of components of $H_k$.
4. The vertices in each slot $c$ at stage $k$ correspond to the vertices in each component of $H_k$.

*Proof.* Throughout the proof, we implicitly use $\mathcal{C}_2$ to ensure that every vertex belongs in exactly 1 slot.

For the first statement, we first note that $\Phi_M(v) = a_v$ is specified by $\mathcal{C}_1$, and the definition of $V_k$ (the active vertex set at stage $k$) specifies that they are examining vertices $v$ with $a_v \leq k$. In particular, we see that $V_k$ and $H_k$ shares the same vertex set.

For the second statement. The number of slots in $cc_k$ is the number $N$ of pairs $(b, d)$ there are such that $b \leq k < d$ from the input PDs. Now we have the equality

$$N = \sum_{b \leq k} \#\{(b, d) \mid d > k\} = \sum_{b \leq k} \sum_{c \in [cc_k]} \mathbf{1}[e_{kcb}],$$

where the second equality follows from $\mathcal{C}_6$. For a fixed $c$ at stage $k$, clearly $e_{kcb}$ can only be 1 for at most one choice of $b$. Thus, we see that

$$\sum_{b\leq k}\sum_{c\in[cc_k]}\mathbf{1}[e_{kcb}]\leq |cc_k| = N$$

with equality if and only if for each $c$, $e_{kcb} = 1$ for some $b$. The fact that $e_{kcb} = 1$ ensures the existence of a vertex in slot $c$ at stage $k$.

For the third statement. Suppose the number of components in $H_k$ is less than $|cc_k|$. Since every slot has to have at least 1 vertex, there exists two vertices $v, w$ in a component of $H_k$ such that $v, w$ belongs to two different slots. By choosing a path if necessary as before, we can without loss pretend that $v, w$ are adjacent to each other. Since $v, w$ are in different slots, there is a slot $c$ such that $x_{kcv} = 1$ and $x_{kcw} = 0$. Then we have that by $\mathcal{C}_3$,

$$M_{vw} = 1 \leq 1 - |x_{kcv} - x_{kcw}| = 1 - |1 - 0| = 0,$$

which is a contradiction.

Now suppose the number of components in $H_k$ is greater than $|cc_k|$. Since each component of $H_k$ has at least 1 vertex, it must be the case that two vertices in different components are assigned to the same slot, which is impossible by $\mathcal{C}_4$. This proves the third statement.

For the fourth statement, by $\mathcal{C}_4$ and (2) we know the vertices in each slot $c$ at stage $k$ is connected and non-empty, so each slot must belong to some component in $H_k$. By $\mathcal{C}_3$ and similar argument as before, this assignment is an injection. By (3), we know this map is a bijection. Thus, we know that the induced subgraph on vertices of $c \in \mathrm{cc}_k$ is a subgraph of a component of $H_k$. The only ambiguity is if $H_k$ contains a vertex $v_+$ not in slot $c$. However, $\mathcal{C}_2$ requires $v_+$ to be in some slot, if it is in any slot other than $c$, the assignment is no longer an injection. This concludes the fourth statement. □

**Recall:** In Definition 2.4, we write $H_\bullet^\Phi$ to denote the sequence of subgraphs specified by the function $\Phi_H$.

Using Lemma F.1 and Lemma F.3, we conclude that:

**Corollary F.4.** $H_k = H_k^{\mathrm{cp}}$ for all $k$. Since the $H_k$'s form the subgraphs in the vertex based filtration function $\Phi_M = \Phi_H$, this shows that

$$\mathcal{P}(\emptyset \subseteq H_0^{\mathrm{cp}} \subseteq ... \subseteq H_n^{cp} = H) = \mathcal{P}(H_\bullet^\Phi).$$

We now prove the soundness of the encoding in Theorem 4.2. (Note the notation change from $H$ to $H(\sigma)$ below)

**Theorem F.5** (Soundness of Encoding)**.** Given a fixed $(\mathcal{P}_0, \mathcal{P}_1)$ from a hidden graph $G$. Let $\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_6\}$ and $\mathcal{F}$ the set of assignment satisfying assignment to $\mathcal{C}$. For every $\sigma \in \mathcal{F}$, the graph $H(\sigma)$ specified has $\mathcal{P}(H(\sigma)_\bullet^\Phi) = (\mathcal{P}_0, \mathcal{P}_1)$.

*Proof.* By Corollary F.4, we have that

$$\mathcal{P}(H(\sigma)_\bullet^\Phi) = \mathcal{P}(H(\sigma)_\bullet^{cp} \coloneqq \emptyset \subseteq H(\sigma)_0^{\mathrm{cp}} \subseteq ... \subseteq H(\sigma)_n^{cp} = H(\sigma)).$$

It suffices to prove the right hand side agrees with $(\mathcal{P}_0, \mathcal{P}_1)$.

For the ease of notation, we will write $\mathcal{P}(\emptyset \subseteq H(\sigma)_0^{\mathrm{cp}} \subseteq ... \subseteq H(\sigma)_n^{cp} = H(\sigma)) = (\mathcal{P}_0^{cn}, \mathcal{P}_1^{cn})$.

**For 1-dimensional PDs:** $\mathcal{C}_5$ ensures that the number of cycles born at each stage $k$ for $H(\sigma)_\bullet^{cp}$ is the same as the number of cycles born at each stage $k$ in $\mathcal{P}_1$.

Note that this also implies $H(\sigma)_k^{cp}$ and $G_k^\Phi$ has the same number of edges. As by construction they already have the same number of vertices and components.

**For 0-dimensional PDs:** It suffices to prove the following: For each fixed $(b, d)$, the number $N(b, d)$ of pairs of the form $(b, d)$ in $\mathcal{P}_0$ and $\mathcal{P}_0^{cn}$ are the same.

For a fixed $b < d$, $N(b, d)$ can be computed as

$N(b, d) = \#\{\text{components born at } b \text{ and still alive at } d-1\} - \#\{\text{components born at } b \text{ and still alive at } d\}$.

This quantity is the same for both sides by $\mathcal{C}_6$.

For $b = d$, $N(d, d)$ can be computed as

$$N(d, d) = \#\{\text{vertex deaths at stage } d\} - \sum_{b<d} N(b, d).$$

Since $H(\sigma)_k^{cp}$ and $G_k^{\Phi}$ has the same number of edges, vertices, components, and cycles for all $k$, we know that both sides have the same number $N$ of vertex deaths that occurred at stage $d$. We established above that the quantity $N(b, d)$ is the same for both sides when $b < d$. Thus, it follows that the quantity $N(d, d)$ is the same for both sides. This concludes the proof. □

Now we will prove completeness of the encoder.

**Theorem F.6** (Completeness of Encoding)**.** Let $K$ be a graph such that $\mathcal{P}(K_\bullet^{\Phi}) = (\mathcal{P}_0, \mathcal{P}_1)$, then there exists an assignment $\sigma \in \mathcal{F}$ such that $H(\sigma) = K$.

*Proof.* Fix an ordering of the vertices of $K$ in the list $[n]$ and an ordering of the components of $K_k^{\Phi}$ at each stage $k$. We give the following assignment to the variables:

1. $M_{ij} = 1$ if $(i, j) \in E(K)$ and $i < j$. $M_{ij} = 0$ if neither $(i, j)$ or $(j, i)$ is in $E(K)$.
2. $x_{kcv} = 1$ if vertex $v$ is in component $c$ at $K_k$ and 0 otherwise. We can fill in all variables because $K_\bullet^{\Phi}$ and $\mathcal{P}_0$ have the same number of components at each stage.
3. We create $\pi_{kcij}$ and $\ell_{kcv}$ by choosing any spanning tree of each component $c$ at $K_k$ coming out of the root node with minimum index.
4. $r_{kcv}$ and $e_{kcb}$ are dependent on the previous variables.

For the precomputed quantity, we have the choice of making an arbitrary bijection, here we associated each $i$ with its value $\Phi_K(i)$. We can do this because the assignment corresponds exactly birth times arising in $\mathcal{P}_0$ by the hypothesis.

We now check that they satisfy the constraints $\mathcal{C}_1, ..., \mathcal{C}_6$ one by one.

1. For $\mathcal{C}_1$: This is automatic from how we chose the bijection.
2. For $\mathcal{C}_2$: A vertex $v$ in $K_k$ cannot belong to more than two components, so it follows that $x_{kcv}$ is 1 for exactly one choice of $c$ at a fixed $(k, v)$. Thus it follows that $\sum_{c\in[cc_k]} x_{kcv} = 1$.
3. For $\mathcal{C}_3$: If $u$ and $v$ does not share an edge, then the inequality is always true. Thus, we consider when $u, v$ shares an edge. By construction here $u, v$ has to be in the same component, so there is a $c \in [cc_k]$ such that $x_{kcu} = x_{kcv} = 1$. Otherwise, for any $c' \neq c$ in $[cc_k]$, we have that $x_{kc'u} = x_{kc'v} = 0$. In either cases we have that $1 \leq 1$.
4. For $\mathcal{C}_4$: Clearly the spanning tree made at each component satisfies the constraints given.
5. For $\mathcal{C}_5$: This follows from a standard fact in algebraic topology (see for instance, Theorem 2.44 of [18]) that for a graph $A$,

   $$\#V(A) - \#E(A) = \#\text{components of } A - \#\text{independent cycles of } A.$$

   This is called the invariance of the Euler characteristics. In particular, $\mathcal{C}_5$ is satisfied by rewriting the equation above to solve for the cycle counts.

6. For $\mathcal{C}_6$: The vertex with minimum filtration value in a component is equal to the birth time of said component. Thus, the count holds for $K_\bullet^\Phi$ as well.

□

Finally, we are able to prove Theorem 4.2. To recall, this is the statement that:

**Theorem.** Let $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_6\}$, $\mathcal{F}$ the set of satisfying assignments to $\mathcal{C}$, and $\mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$ the set of graphs $H$ with $\mathcal{P}(H_\bullet^\Phi) = (\mathcal{P}_0, \mathcal{P}_1)$. Define $H(\sigma)$ from $\sigma \in \mathcal{F}$ by $E(H(\sigma)) = \{\{i, j\} : \sigma(M_{ij}) = 1\}$. Then:

1. **Soundness:** $H(\sigma) \in \mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$ for every $\sigma \in \mathcal{F}$.
2. **Completeness:** every $H \in \mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$ admits at least one $\sigma \in \mathcal{F}$ with $H(\sigma) = H$.
3. **Minimality for Degree:** When $\Phi = \deg$, for each $i \in \{1, \ldots, 6\}$, there exist a target $(\mathcal{P}_0, \mathcal{P}_1)$ and an assignment $\sigma$ that satisfies $\mathcal{C} \setminus \{\mathcal{C}_i\}$ but with $H(\sigma) \notin \mathcal{H}(\mathcal{P}_0, \mathcal{P}_1)$.

*Proof of Theorem 4.2.* Soundness is proven in Theorem F.5, and completeness is proven in Theorem F.6. For minimality of degree, we removed each of the six constraints and generated an assignment that does not satisfy the removed constraints. The counter-examples are written in Appendix G and explained there. □

## G Per-Constraint Counterexamples for Degree-Based PD

For each necessary constraint $\mathcal{C}_i$, we exhibit the a target $G$ we found in our search where, with $\mathcal{C}_i$ disabled, the constraint solver returns a graph $H$ with $\mathrm{PH}(H) \neq \mathrm{PH}(G)$. Throughout this section, we will always use degree-based filtration and consider degree-based persistence diagrams.

Counterexamples were found by enumerating up to 80 distinct CP solutions per (target, dropped-constraint) pair across all named graphs on $|V| \leq 6$ and many random samples for $|V| \leq 9$. Figure 4 displays the target graph and the counterexample produced.

In this section, we use $(b, d)^n$ to indicate the tuple $(b, d)$ has multiplicity $n$.

**Remark G.1.** Before we proceed, we make a comment on how the $\mathcal{C}_1$ constraint is verified. Recall that in the actual generation process, the underlying graph $G$ is hidden from us, so we only have access to the persistence tuples. Therefore, $\mathcal{C}_1$ is satisfied as long as one can match the output $\{\Phi_H(v), v \in H\}$ to the multi-set of birth times that we could read off from $\mathcal{P}_0$. In particular, for degree, this means that the degree of the literal index $i$ in $H$ need not be the degree of the literal index $i$ in $G$. Of course, after reordering if necessary, we can make this true.

### G.1 $\mathcal{C}_1$ (degree)

**Target.** $G = 2K_2$: $V = \{0, 1, 2, 3\}$, $E_G = \{(0, 1), (2, 3)\}$. Degrees $(1, 1, 1, 1)$. $\mathcal{P}_0(G) = \{(1, 1)^2, (1, \infty)^2\}$, $\mathcal{P}_1(G) = \emptyset$.

**Counterexample.** $H$: $E_H = \{(0, 1), (0, 3)\}$.

**Witness.**

$$M_{01} = M_{03} = 1, \quad M_{ij} = 0 \text{ otherwise}$$
$$x_{1,0,0} = x_{1,0,1} = x_{1,0,3} = 1, \quad x_{1,1,2} = 1$$
$$\pi_{1,0,0,1} = \pi_{1,0,0,3} = 1, \quad \ell_{1,0,0} = 0,\ \ell_{1,0,1} = \ell_{1,0,3} = 1.$$

**Verification.**

- $\mathcal{C}_2$: each vertex in one slot.
- $\mathcal{C}_3$: edges $(0, 1), (0, 3)$ within slot 0.
- $\mathcal{C}_4$: slot 0 spans the star $K_{1,2}$.
- $\mathcal{C}_5$: $2 - 4 + 2 = 0 = \mathrm{cy}_1$.
- $\mathcal{C}_6$: $\sum_c \mathbf{1}[e_{1,c,1}] = 2 = \alpha(1, 1)$.

**Why $\mathcal{C}_1$ fails.** $\deg_H(0) = 2$, $\deg_H(2) = 0$.

**PH Comparison.** $\mathrm{PH}(H) = (\{(0, \infty), (1, 2), (1, \infty), (2, 2)\}, \emptyset) \neq \mathrm{PH}(G)$.

### G.2 $\mathcal{C}_2$ (partition)

**Target.** $G = P_6$: $E_G = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)\}$, degrees $(1, 2, 2, 2, 2, 1)$. $\mathcal{P}_0(G) = \{(1, 2), (1, \infty), (2, 2)^4\}$, $\mathcal{P}_1(G) = \emptyset$. Decoded: $cc_1 = 2$, $cc_2 = 1$, $\alpha(1, 1) = 2$, $\alpha(2, 1) = 1$, $\alpha(2, 2) = 0$.

**Counterexample.** $H = P_3 \sqcup K_3$: $E_H = \{(0, 5), (1, 5), (2, 3), (2, 4), (3, 4)\}$. This is the disjoint union of the path graph 0-5-1 and a triangle on $\{2, 3, 4\}$.

**Witness (relaxed partition; vertices $\{2, 3, 4\}$ assigned to no slot at $k = 2$).**

$$M_{05} = M_{15} = M_{23} = M_{24} = M_{34} = 1$$
$$x_{1,0,0} = 1,\ x_{1,1,1} = 1$$
$$x_{2,0,0} = x_{2,0,1} = x_{2,0,5} = 1, \quad x_{2,0,2} = x_{2,0,3} = x_{2,0,4} = 0$$
$$\pi_{2,0,0,5} = \pi_{2,0,5,1} = 1, \quad \ell_{2,0,0} = 0,\ \ell_{2,0,5} = 1,\ \ell_{2,0,1} = 2.$$

**Verification.**

- $\mathcal{C}_1$: the multi-set of degrees agree.
- $\mathcal{C}_3$: edges $(0,5),(1,5)$ in slot 0; $(2,3),(2,4),(3,4)$ outside any slot.
- $\mathcal{C}_4$: slot 0 spans path 0-5-1.
- $\mathcal{C}_5$: $|E_2| - |V_2| + cc_2 = 5 - 6 + 1 = 0 = \mathrm{cy}_2$.
- $\mathcal{C}_6$: slot 0 elder-of-1 (contains $0, 1$, no younger); not elder-of-2 (contains 5 but also younger). Sums match $\alpha(2,1) = 1$, $\alpha(2,2) = 0$.

The up-shot here is that since vertices $2, 3, 4$ are not registered in the component, the pass for $\mathcal{C}_5$ and $\mathcal{C}_6$ were only conducted on the path graph $0 - 5 - 1$. Thus, the assignment was satisfied.

**Why $\mathcal{C}_2$ fails.** $\sum_c x_{2,c,2} = 0 \neq 1$, similarly for vertices 3 and 4.

**PH Comparison.** $\mathrm{PH}(H) = (\{(1,2),(1,\infty),(2,2)^3,(2,\infty)\},\{(2,\infty)\}) \neq \mathrm{PH}(G)$.

### G.3 $\mathcal{C}_3$ (edge-slot consistency)

**Target.** $G = 2K_3$: $E_G = \{(0,1),(0,2),(1,2),(3,4),(3,5),(4,5)\}$, all degrees 2. $\mathcal{P}_0(G) = \{(2,2)^4,(2,\infty)^2\}$, $\mathcal{P}_1(G) = \{(2,\infty)^2\}$.

**Counterexample.** $H \cong C_6$: $E_H = \{(0,1),(0,2),(1,4),(2,3),(3,5),(4,5)\}$.

**Witness (partition respected, cross-slot edges allowed).**

$$\begin{aligned}
&\text{slot } 0 = \{0,1,2\} : x_{2,0,0} = x_{2,0,1} = x_{2,0,2} = 1\\
&\text{slot } 1 = \{3,4,5\} : x_{2,1,3} = x_{2,1,4} = x_{2,1,5} = 1\\
&\pi_{2,0,0,1} = \pi_{2,0,0,2} = 1, \quad \ell_{2,0,0} = 0,\ \ell_{2,0,1} = \ell_{2,0,2} = 1\\
&\pi_{2,1,3,5} = \pi_{2,1,5,4} = 1, \quad \ell_{2,1,3} = 0,\ \ell_{2,1,5} = 1,\ \ell_{2,1,4} = 2.
\end{aligned}$$

**Verification.**

- $\mathcal{C}_1$: the multi-set of degrees agree.
- $\mathcal{C}_2$: clear.
- $\mathcal{C}_4$: each slot uses 2 in-slot edges to span a path.
- $\mathcal{C}_5$: $6 - 6 + 2 = 2 = \mathrm{cy}_2$.
- $\mathcal{C}_6$: each slot elder-of-2; sum $= 2 = \alpha(2,2)$.

The up-shot here is that topologically the graph constructed has one component, but slot-wise there are still two slots going on. The calculations for $\mathcal{C}_5$ and $\mathcal{C}_6$ are conducted using slots rather than components, hence the assignment was satisfied.

**Why $\mathcal{C}_3$ fails.** Edges $(1,4)$ and $(2,3)$ have endpoints in different slots.

**PH Comparison.** $\mathrm{PH}(H = C_6) = (\{(2,2)^5,(2,\infty)\},\{(2,\infty)\}) \neq \mathrm{PH}(G)$.

### G.4 $\mathcal{C}_4$ (slot connectivity)

**Target.** $G = C_6$: $E_G = \{(0,1),(0,5),(1,2),(2,3),(3,4),(4,5)\}$. $\mathcal{P}_0(G) = \{(2,2)^5,(2,\infty)\}$, $\mathcal{P}_1(G) = \{(2,\infty)\}$.

**Counterexample.** $H = 2K_3$: $E_H = \{(0,3),(0,4),(1,2),(1,5),(2,5),(3,4)\}$.

**Witness (single slot covering disconnected graph).**

$$M \text{ encodes } H, \quad x_{2,0,v} = 1\ \forall v.$$

Spanning-tree variables $\pi, \ell$ are unconstrained (assign all zero).

**Verification.**

- $\mathcal{C}_1$: the multi-set of degrees agree.

- $\mathcal{C}_2$: every vertex is in slot 0.
- $\mathcal{C}_3$: all edges within slot 0.
- $\mathcal{C}_6$: $6 - 6 + 1 = 1 = \mathrm{cy}_2$.
- $\mathcal{C}_7$: slot 0 elder-of-2; sum $= 1 = \alpha(2, 2)$.

**Why $\mathcal{C}_4$ fails.** Slot 0's induced subgraph in $H$ is two disjoint triangles (so the subgraph is disconnected). No assignment of $\pi, \ell$ satisfies the spanning-tree depth constraint.

**PH Comparison.** $\mathrm{PH}(H = 2K_3) = (\{(2, 2)^4, (2, \infty)^2\}, \{(2, \infty)^2\}) \neq \mathrm{PH}(G)$.

Before we move on to $\mathcal{C}_5$ and $\mathcal{C}_6$.

### G.5 $\mathcal{C}_5$ (Euler characteristics)

We searched all named graphs and 200+ Erdős–Rényi (ER) random graph samples on $|V| \leq 7$ without finding a graph-level counterexample. The smallest in our search appears at $|V| = 8$.

**Target.** $G = \mathrm{ER}(8, 0.4)$ with seed 5:

$$E_G = \{(0, 7), (1, 6), (2, 3), (2, 6), (2, 7), (3, 4), (3, 7), (4, 6), (5, 6), (5, 7)\}.$$

Degrees are $(1, 1, 4, 3, 2, 2, 4, 4)$. $\mathcal{P}_0(G) = \{(1, 4), (1, \infty), (2, 4)^2, (3, 3)^2, (4, 4)^2\}$, $\mathcal{P}_1(G) = \{(4, \infty)^3\}$.

**Counterexample.** $H$:

$$E_H = \{(0, 6), (1, 6), (2, 4), (2, 5), (3, 6), (3, 7), (4, 5), (4, 7), (5, 7), (6, 7)\}.$$

**Witness.** The graph $H$ and its associated degree-based filtration specifies the $M_{ij}$'s and $x_{kcv}$'s. The spanning tree variables are done at each component. The following table records the progression of relevant data in $G$ and $H$ as the stages $k$ progress.

| $k$ | $\lvert V_k\rvert$ | $\lvert E_k(G)\rvert$ | $\lvert E_k(H)\rvert$ | $cc_k$ | $\mathrm{cy}_k(G)$ | $\mathrm{cy}_k(H)$ |
|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 0 | 2 | 0 | 0 |
| 2 | 4 | 0 | 0 | 4 | 0 | 0 |
| **3** | **6** | **2** | **3** | **4** | **0** | **1** |
| 4 | 8 | 10 | 10 | 1 | 3 | 3 |

**Why $\mathcal{C}_5$ Failed.** Both graphs share the same per-level component structure and birth-time survival. A witness assigning each vertex to its component's slot at every level satisfies $\mathcal{C}_1$-$\mathcal{C}_4, \mathcal{C}_6$. Only $\mathcal{C}_5$'s equation

$$|E_k| - |V_k| + cc_k = \mathrm{cy}_k$$

distinguishes the two graphs at $k = 3$:

**PH Comparison.** Here we have that $\mathcal{P}_0(H) = \mathcal{P}_0(G)$, but $\mathcal{P}_1(H) = \{(3, \infty), (4, \infty)^2\} \neq \mathcal{P}_1(G) = \{(4, \infty)^3\}$.

### G.6 $\mathcal{C}_6$ (birth-time survival)

**Target** $G = \mathrm{ER}(7, 0.3)$ with seed 28:

$$E_G = \{(0, 1), (0, 2), (0, 4), (0, 5), (1, 3), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (4, 5)\}.$$

Degrees are $(4, 3, 5, 2, 3, 3, 2)$. $\mathcal{P}_0(G) = \{(2, 3), (2, \infty), (3, 3)^2, (3, 4), (4, 4), (5, 5)\}$, $\mathcal{P}_1(G) = \{(4, \infty), (5, \infty)^4\}$.

**Counterexample.** $H$:

$$E_H = \{(0, 2), (0, 6), (1, 5), (1, 6), (2, 3), (2, 6), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6)\}.$$

**Witness.** The graph $H$ and its associated degree-based filtration specifies the $M_{ij}$'s and $x_{kcv}$'s. The spanning tree variables are done at each component. At every level, $|V_k|$, $|E_k|$, $cc_k$, and $\mathrm{cy}_k$ are *identical* between $G$ and $H$:

| $k$ | $\lvert V_k\rvert$ | $\lvert E_k\rvert$ | $cc_k$ | $\mathrm{cy}_k$ |
|---|---|---|---|---|
| 2 | 2 | 0 | 2 | 0 |
| **3** | **5** | **3** | **2** | **0** |
| 4 | 6 | 6 | 1 | 1 |
| 5 | 7 | 11 | 1 | 5 |

**Why $\mathcal{C}_6$ Failed.** The difference occurs in $\alpha(k, b)$ at $k = 3$. Recall $\alpha(3, 2)$ refers to the number of components born at 2 that is still alive at stage 3.

| | $\alpha(3,2)$ |
|---|---|
| $G$ | 1 |
| $H$ | 2 |

**PH Comparison.** $\mathcal{P}_0(H) = \{(2, 4), (2, \infty), (3, 3)^3, (4, 4), (5, 5)\} \neq \mathcal{P}_0(G)$, and $\mathcal{P}_1(H) = \mathcal{P}_1(G)$.

### G.7 $\mathcal{C}_7$ (ancestry)

For completeness, we also empirically tested adding vs. not adding the redundant constraint $\mathcal{C}_7$. Across all 95 test cases (graphs up to $|V| = 30$), disabling $\mathcal{C}_5$ yields no graph-level counterexample. Moreover, the solver runs $1.42\times$ faster without $\mathcal{C}_5$ (23.0 s vs. 16.2 s aggregate).

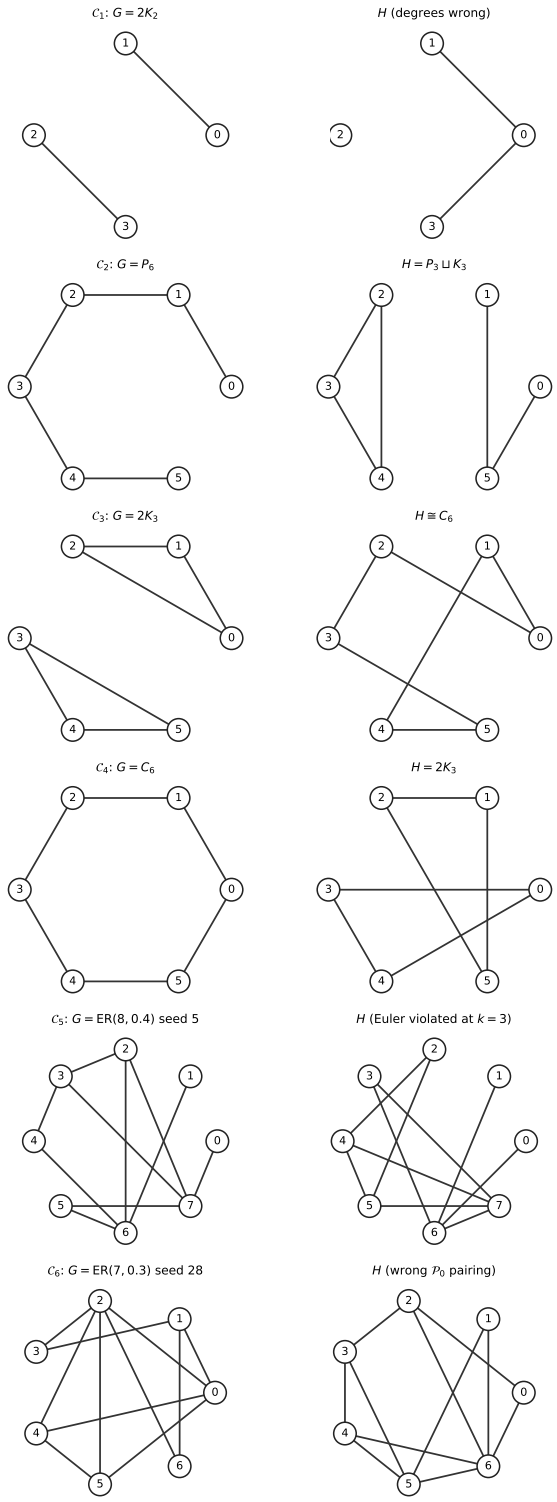



Figure 4: Counterexamples in Appendix G. The left displays the target graph and the right displays a graph generated with constraint $\mathcal{C}_i$ removed that has a different degree PD.

# H Experimental Details

## H.1 Datasets

We evaluate on four standard graph generation benchmarks: COMM-20 (small dense two-community SBMs), PLANAR (fixed-size random planar graphs), SBM (large stochastic block model graphs), and ENZYMES (real-world enzyme protein contact maps from the TUDataset [34]). The first three are synthetic and follow the conventions of prior graph generation work [32, 52, 40, 30]; ENZYMES is a real-world dataset also previously used in prior work. We adopt the uniform $64/16/20\%$ train/validation/test split convention from SPECTRE [32].

Aggregate statistics are listed in Table 4. The four datasets cover a range of regimes that stress different aspects of our methods. COMM-20 is the smallest and densest, with the lowest max degree; the easiest setting in which to demonstrate strict PD compliance. PLANAR has fixed $|V| = 64$ and every graph is guaranteed to be connected and planar by construction. SBM drives the difficulty: largest graphs ($|V|$ up to 187), highest edge counts ($|E|$ up to 1129), and highest max degree ($\bar{\Delta} = 17.1$), and is where constraint-programming scalability is most stressed. ENZYMES tests generalisation beyond synthetic benchmarks.

Table 4: Dataset statistics (all splits combined). Splits follow the uniform $64/16/20\%$ convention.

| | COMM-20 | PLANAR | SBM | ENZYMES |
|---|---|---|---|---|
| Total graphs | 100 | 200 | 200 | 587 |
| Splits (train / val / test) | 64 / 16 / 20 | 128 / 32 / 40 | 128 / 32 / 40 | 376 / 94 / 117 |
| Nodes (min / mean / max) | 12 / 15.7 / 20 | 64 / 64 / 64 | 44 / 104.0 / 187 | 10 / 33.0 / 125 |
| Edges (min / mean / max) | 21 / 37.0 / 61 | 173 / 177.8 / 181 | 129 / 500.4 / 1129 | 16 / 63.3 / 149 |
| Max. degree (mean) | 6.5 | 9.0 | 17.1 | 6.1 |

## H.2 Baseline Methods

We compare against four neural baselines along two axes: the diffusion backbone (DIGRESS or CONSTRUCT) and whether the topology-aware components of TAGG (TAM module and PDM loss) are added. This yields DIGRESS, CONSTRUCT, DIGRESS+TAGG, and CONSTRUCT+TAGG.

### H.2.1 DiGress

DIGRESS [52] is a discrete denoising diffusion model and the foundation for both CONSTRUCT and TAGG. The forward process independently corrupts each node and edge category through transition matrices $Q_X^t, Q_E^t$. The reverse process is a graph transformer $\phi_\theta$ that maps a noisy graph $G^t$ to predicted distributions $(\hat{p}^X, \hat{p}^E)$ over clean-graph categories, trained with cross-entropy

$$\mathcal{L}_{\mathrm{CE}}(G_0, \hat{p}^G) = \sum_i \mathrm{CE}(x_i, \hat{p}_i^X) + \lambda \sum_{i,j} \mathrm{CE}(e_{ij}, \hat{p}_{ij}^E).$$

We use the marginal-noise variant, which sets the limit distribution to the empirical node and edge marginals of the training set, and augment the denoising network with auxiliary cycle counts and Laplacian spectral features to mitigate the expressivity limits of message passing [60]. DIGRESS is unconstrained and serves as the vanilla diffusion baseline.

### H.2.2 ConStruct

CONSTRUCT [30] extends DIGRESS with hard constraints for *edge-deletion-invariant* properties (planarity, acyclicity, max degree, etc.). It introduces two modifications.

**Edge-absorbing noise model.** The forward edge transition sends each edge either to itself or to an absorbing no-edge state,

$$Q_E^t = \alpha_{\mathrm{abs}}^t I + (1 - \alpha_{\mathrm{abs}}^t)\, \mathbf{1}_c\, e'_E,$$

making the forward sequence a chain of subgraphs $G^0 \supseteq G^1 \supseteq \cdots \supseteq G^T = \emptyset$ and the reverse process pure edge insertion.

**Connection to graph filtrations.** The forward chain $G^0 \supseteq G^1 \supseteq \cdots \supseteq G^T = \emptyset$ (and its reverse) is structurally a general filtration of $G^0$ in the sense of Sec. 3.1, with the edge-insertion order set by the diffusion schedule rather than a canonical filter function. CONSTRUCT is not topology-aware and the schedule has no relation to persistent homology. Still, the subgraph-chain skeleton aligns more naturally with the filtration setting of persistent homology than DIGRESS's independent-edge corruption does, making CONSTRUCT a more filtration-compatible baseline even before any topological signal is added on top.

**Projector.** At each reverse step, given the candidate $\hat{G}^{t-1}$ and previous iterate $G^t$, the projector iteratively inserts candidate edges $E(\hat{G}^{t-1}) \setminus E(G^t)$ in random order, discarding any whose insertion violates the target property. Theorem 1 of [30] shows this recovers an optimal solution to $\min_{G \in \mathcal{C}} \mathrm{GED}(\hat{G}^{t-1}, G)$ for $\mathcal{C} = \{G : \mathcal{P}(G), G \supseteq G^t\}$.

We apply CONSTRUCT with planarity on PLANAR; for the other three datasets, which lack a uniform edge-deletion-invariant property, we run the same architecture with the projector disabled.

#### H.2.3 TAGG: Topology-Aware Attention and PDM Loss

TAGG [40] uses the degree-based filtration as a conditioning signal and proposes two topology-aware components built on top of it, both of which we re-implement and apply to both CONSTRUCT and DIGRESS backbones.

**PDM loss.** The persistence diagrams $D_{G_0}, D_{\hat{p}^{G_0}}$ are computed under the degree-based filtration (matching our setting up to normalisation). The loss is the 1-Wasserstein distance with diagonal padding for cardinality mismatch and per-dimension matching to prevent $\mathcal{P}_0$/$\mathcal{P}_1$ confusion:

$$\mathcal{L}_{\mathrm{PDM}}(G_0, \hat{p}^{G_0}) = W_1(D_{G_0}, D_{\hat{p}^{G_0}}).$$

The full objective is $\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \alpha\, \mathcal{L}_{\mathrm{PDM}}$.

**Topology-aware attention (TAM).** A homological feature vector $\mu_{G_0} \in \mathbb{R}^{L \cdot S}$, derived from the persistence landscape of $G_0$, is injected as a global bias in attention scores at every denoising layer:

$$A_t^m = \mathrm{softmax}\Big(\tfrac{Q_t^m (K_t^m)^\top}{\sqrt{d_k}} \cdot Z_{E_t}^m + H_{G_0}\Big),$$

with $H_{G_0} = \mathrm{Lin}_\mu(\mu_{G_0})$. At inference, $G_0$ is unknown, and the original TAGG substitutes the training-set average $\mu_{G'}$ as conditioning, making it a distribution-level method rather than a per-target one. To align it with our setting — where the target PD is given per sample — we instead feed the homological feature vector $\mu_{G_{\mathrm{test}}}$ derived from each test graph as the conditioning signal at inference. This grants TAGG the strongest possible per-target topology signal, putting it on equal footing with our methods for the strict PD-compliance evaluation.

#### H.2.4 Augmenting DiGress and ConStruct with TAGG

DIGRESS+TAGG adds both TAM and the PDM loss to DIGRESS, recovering the original TAGG configuration. CONSTRUCT+TAGG adds the same components on top of CONSTRUCT's edge-absorbing forward process and projector-augmented reverse process; the projector remains active and filters edges proposed by the TAGG-augmented denoiser. In both cases, inference uses the per-target $\mu_{G_{\mathrm{test}}}$ derived from each test graph, as described above.

**Hyperparameter selection.** For each baseline we use the default hyperparameters proposed in the original paper for the datasets considered.

### H.3 Hardware

The constraint-programming solver and the local rejection sampler ran on a 120-CPU-core machine; no GPU was used. Neural diffusion baselines were trained on $4\times$ NVIDIA A100 (40 GB) GPUs and run at inference on a single A100 or RTX 6000 Ada (48 GB).

### H.4 Local vs. Global: Full Comparison Across Filtrations

We expand Sec. 5.1 with the full set of local generators across the three filtration regimes considered in Sec. 3, together with median statistics and per-method commentary.

**Methods compared.** Recall from Sec. 3 that the local approach yields three rejection samplers, each tied to a different filtration scheme:

- RANDOM (Algorithm 1): general filtration. The vertex and edge filter values $f_V, f_E$ are not canonical and are sampled along with the graph.
- VERTEX (Algorithm 2): vertex-based filtration. Edges are forced to take time $\max(f_V(u), f_V(v))$, but $f_V$ itself is not canonical and is sampled.
- DEGREE (Algorithm 3): degree-based filtration. Both vertex and edge filter values are uniquely determined by the graph, so the algorithm only needs to return the graph itself.

For RANDOM and VERTEX, the algorithm therefore returns the pair $(H, f)$ specifying both the graph and a filtration function realising the target PD; only DEGREE (and the global CP solver) work directly with the canonical degree filter $f_V(v) = \deg(v)$. In this comparison the CP solver targets the degree filtration, so that it is compared with DEGREE on the same canonical filtration. Other filtrations are verified in Appendix J.

**Remark H.1.** In the implementation of Algorithm 2 and Algorithm 3, we took an alternative approach to the **minimum vertex pivot** step.

In any stage of vertex based filtrations, let $b_{\min} = \inf_{\text{component deaths of the form } (b,k+1)} b$. In the minimum vertex pivot step, we instead do the following procedure:

1. If there is only one component born at $b_{\min}$, randomly pick a component born at $t < b_{\min}$ and connect to a new vertex.
2. Otherwise, randomly pick a component born at $t \leq b_{min}$ and connect to a new vertex.

If the algorithm terminates, this is still guaranteed to give a PD-equivalent output and could realize all possible PD-equivalent vertex-based filtrations. This choice involved less variables to check, but a caveat is that this could potentially fail the component merging step.

**Setup.** We follow the protocol of Sec. 5.1: each rejection sampler is allowed up to $20{,}000$ proposals per target with $4$ parallel evaluator threads, and CP is given a $1{,}800$ s budget with $120$ internal CP-SAT search workers. Wall-clocks are measured on a clean 120-CPU-core machine. We report success rate, mean and median per-target wall-clock (charging the full budget on failed runs), and mean and median attempts to first success on successful runs.

**Results.** Table 5 reports the full set of numbers.

**Discussion.** The three local samplers form a clean ordering from least to most constrained. RANDOM, working under a general filtration where $f_V, f_E$ are jointly sampled, accepts almost every proposal (mean attempts 1–3 across all four datasets) and finishes in seconds. VERTEX, with edges deterministically pinned to $\max(f_V(u), f_V(v))$, sees attempt counts grow to a few hundred on the larger benchmarks but still completes within $9$ s on every dataset. DEGREE, where the filtration is fully canonical and the PD effectively determines the degree sequence, is where rejection becomes expensive: the per-attempt success probability collapses on large dense instances, and on SBM only $15\%$ of test targets are solved within the $20{,}000$-attempt budget. The CP solver targets the same degree filtration but, by enforcing the global constraints up front, raises the success rate to $82.5\%$ on SBM and cuts the mean wall-clock by more than a factor of two. On COMM-20, PLANAR, and ENZYMES the two degree-filtration methods solve essentially all targets, with the local sampler $1.7$–$2.8\times$ faster on the smaller datasets and on par on ENZYMES.

**Tail behaviour.** The mean–median gap on SBM and ENZYMES reflects a heavy right tail in attempt counts. For DEGREE/ENZYMES (where success is $99.1\%$), $50\%$ of targets are solved in $\leq 15$ attempts, but the top $5\%$ require $\geq 441$ and the worst $17{,}331$. The mean ($431.5$) is dominated by this tail and overstates the typical cost. This is the natural pattern of rejection sampling against PD constraints derived from heterogeneous graphs: most reference PDs are typical and easy to match by chance, but a few have rare structure that the proposal distribution rarely hits.

**Practical recipe.** The pair (success rate, mean attempts) characterises the difficulty of each (filtration, dataset) cell. Mean attempts $\leq 100$ marks an easy cell where the local sampler is the right tool; $\geq 1{,}000$ a borderline cell; $\geq 10{,}000$ (or hitting `max_attempts`) a hard cell where CP becomes necessary. A simple hybrid — run the local sampler with a modest cap (e.g. $1{,}000$ attempts) and fall back to CP on rejection — achieves the best of both: free generation on the easy majority of targets,

Table 5: Full local/global comparison across the three filtration regimes. RANDOM and VERTEX target their respective filtrations; DEGREE (Alg. 3) and CP both target the canonical degree filtration. Wall-clock means include the full 20,000-attempt or 1,800 s budget on failed runs; attempt statistics are over successful runs only. CP performs a single solver call per target.

| | | COMM-20 | PLANAR | SBM | ENZYMES |
|---|---|---|---|---|---|
| RANDOM (Alg. 1) | Success (%) ↑ | 100.0 | 100.0 | 100.0 | 100.0 |
| | Wall-clock mean (s) ↓ | 0.10 | 2.02 | 6.04 | 0.71 |
| | Wall-clock median (s) ↓ | 0.09 | 2.02 | 5.30 | 0.65 |
| | Attempts mean ↓ | 1.0 | 2.8 | 1.1 | 1.7 |
| | Attempts median ↓ | 1 | 1 | 1 | 1 |
| VERTEX (Alg. 2) | Success (%) ↑ | 100.0 | 100.0 | 100.0 | 100.0 |
| | Wall-clock mean (s) ↓ | 0.11 | 2.82 | 8.70 | 0.86 |
| | Wall-clock median (s) ↓ | 0.10 | 2.82 | 6.31 | 0.76 |
| | Attempts mean ↓ | 1.1 | 258.6 | 476.6 | 81.0 |
| | Attempts median ↓ | 1 | 185.5 | 231.0 | 41.0 |
| DEGREE (Alg. 3) | Success (%) ↑ | 100.0 | 100.0 | **15.0** | 99.1 |
| | Wall-clock mean (s) ↓ | 0.17 | 5.88 | 1465.0 | 6.94 |
| | Wall-clock median (s) ↓ | 0.13 | 4.62 | $\sim$3000$^\dagger$ | 1.07 |
| | Attempts mean ↓ | 49.6 | 137.5 | 4583.5 | 431.5 |
| | Attempts median ↓ | 6 | 70.0 | 3552.5 | 15 |
| CP (Sec. 4) | Success (%) ↑ | 100.0 | 100.0 | **82.5** | 100.0 |
| | Wall-clock mean (s) ↓ | 0.30 | 16.35 | 654.8 | 6.85 |
| | Wall-clock median (s) ↓ | 0.16 | 16.21 | 142.5 | 1.78 |

$^\dagger$ DEGREE/SBM: only $6/40$ targets succeed; the median is dominated by the failure budget. Successful runs alone have a median wall-clock of 162 s.

principled fallback on the hard tail. We use the local-only and CP-only configurations in the main paper for clarity of presentation.

### H.5 Neural Baseline Comparison: Full Details

This appendix expands Sec. 5.2 with full descriptions of the training and inference protocols, the evaluation metric, the full results table with standard deviations (Table 6, the unabridged version of Table 3), and discussion of each (method, dataset) cell.

**Inference protocol: paired vs. averaged conditioning.** The original TAGG [40] substitutes the dataset-averaged landscape vector $\mu_{G'} = \frac{1}{|D_{\text{train}}|} \sum_{G \in D_{\text{train}}} \mu_G$ as the conditioning signal at every test sample, since the test graph $G_0$ is unknown at inference. This makes TAGG a distribution-level method: it generates graphs whose topology resembles the training-set average, not a specific target. To align it with our setting — where the target PD is given per sample — we instead feed each test graph's own $\mu_{G_{\text{test}}}$ at inference. Concretely, in the test loop we iterate the test data loader and pass each batch's $\mu$ vector and $n$-nodes mask into the reverse-diffusion sampling routine, producing one generated graph per test reference. This change has two consequences: (i) it grants the TAGG variants the strongest possible per-target topology signal, putting them on equal footing with our CP solver for the strict PD-compliance evaluation; (ii) it yields a 1-to-1 correspondence between generated and reference graphs, enabling the paired $\bar{W}_1$ metric. Within our paper, all three methods (CP, CONSTRUCT+TAGG, DIGRESS+TAGG) are evaluated under this paired protocol so they are pairwise comparable; absolute $\bar{W}_1$ numbers are not directly comparable to the original TAGG paper's reported numbers, which use the averaged $\mu_{G'}$ protocol.

**Evaluation metric details.** For each test reference $G_{\text{ref}}^{(i)}$ paired with its generated graph $H^{(i)}$, we compute persistence diagrams under the degree filtration using `gudhi` and pair them via the standard 1-Wasserstein distance with diagonal padding for unequal-cardinality multisets. The two homology dimensions are summed:

$$\bar{W}_1^{(i)} \;=\; W_1\big(\mathcal{P}_0(G_{\text{ref}}^{(i)}), \mathcal{P}_0(H^{(i)})\big) + W_1\big(\mathcal{P}_1(G_{\text{ref}}^{(i)}), \mathcal{P}_1(H^{(i)})\big),$$

and we aggregate as $\bar{W}_1 = \frac{1}{N_{\text{test}}} \sum_i \bar{W}_1^{(i)}$. The CP solver's $\bar{W}_1 = 0.000$ entries serve as a sanity check: they confirm both that CP's output PD matches the target exactly and that our $W_1$ measurement code returns zero on identical multisets (no float-precision dust). The numerical scale of $\bar{W}_1$ varies across datasets because larger graphs have larger PDs — absolute magnitudes between datasets are not directly comparable, only relative orderings within a column.

**Hardware and budgets.** CP is run on a 120-core CPU slice with `cp_sat_workers=120` and a 1,800 s per-target time limit. Neural inference runs on a single GPU (RTX 6000 Ada or A6000) per cell, single-process. Training is multi-GPU when available (typically 4 GPUs with DDP). The trained checkpoint per cell is selected as the lowest-validation-MMD-mean checkpoint among the top-3 lowest `val/epoch_NLL` checkpoints, automated in the test branch of each `train_*.py` script.

**Full results with standard deviations.** Table 6 reports the unabridged version of Table 3, with neural variants summarised as mean ± std over 5 random seeds. CP is deterministic and reported as a single number. Standard deviations are uniformly small (1–7% of the mean), confirming that the residual gap between neural variants and CP is structural rather than due to seed noise.

Table 6: Full version of Table 3: GLOBAL (CP) vs. TAGG-augmented neural baselines, with standard deviations over 5 random seeds. CP is deterministic.

| | | COMM-20 | PLANAR | SBM | ENZYMES |
|---|---|---|---|---|---|
| GLOBAL (CP) | $\bar{W}_1 \downarrow$ | **0.000** | **0.000** | **0.000** | **0.000** |
| | Time/graph (s) $\downarrow$ | 0.30 | 16.35 | 654.8 | 6.85 |
| | Train (h) $\downarrow$ | 0 | 0 | 0 | 0 |
| CONSTRUCT+TAGG | $\bar{W}_1 \downarrow$ | $9.35 \pm 0.97$ | $43.23 \pm 0.56$ | $434.48 \pm 17.54$ | $17.35 \pm 0.24$ |
| | Time/graph (s) $\downarrow$ | $0.88 \pm 0.01$ | $4.73 \pm 0.09$ | $17.11 \pm 0.09$ | $4.92 \pm 0.46$ |
| | Train (h) $\downarrow$ | 9.4 | 24.3 | 59.8 | 61.6 |
| DIGRESS+TAGG | $\bar{W}_1 \downarrow$ | $4.52 \pm 0.32$ | $38.77 \pm 1.07$ | $464.99 \pm 32.73$ | $17.06 \pm 0.44$ |
| | Time/graph (s) $\downarrow$ | $0.90 \pm 0.00$ | $5.30 \pm 0.28$ | $23.16 \pm 0.47$ | $5.94 \pm 0.05$ |
| | Train (h) $\downarrow$ | 8.9 | 32.5 | 47.6 | 49.2 |

Table 7: Handling of failed CP runs on SBM. CP solves $33/40$ targets within the 1,800 s budget (the coverage reported in Table 2); its $\bar{W}_1$ in Table 3 is over these solved targets. Restricting the neural baselines to the same 33 targets gives an equal comparison.

| SBM, $\bar{W}_1 \downarrow$ | All 40 targets | Same 33 targets (CP-solved) |
|---|---|---|
| CONSTRUCT+TAGG | 434.48 | 408.95 |
| DIGRESS+TAGG | 464.99 | 454.49 |
| CP (ours) | 0.00 ($33/40$ solved) | 0.00 |

**Per-dataset observations.**

- **COMM-20.** The smallest and densest dataset. CP solves each target in 0.3 s — the constraint problem is small enough that solver overhead dominates — and is the only cell where CP is faster than the neural variants. DIGRESS+TAGG achieves the lowest neural $\bar{W}_1$ (4.52), roughly halving CONSTRUCT+TAGG's (9.35); the marginal-distribution transition seems to give a more PD-faithful denoising path on small graphs.
- **PLANAR.** The fixed-size benchmark. CP scales to 16.35 s/graph, 3–3.5× slower than the neural variants, since the cycle structure under degree filtration ($\sim$110 cycles, max degree 8–12) generates a moderately large slot/ancestry encoding. The two TAGG variants achieve similar $\bar{W}_1$ ($\sim$40).
- **SBM.** The hardest cell. $7/40$ CP runs hit the 1,800 s timeout, contributing the worst-case 1,800 s to the mean (654.8 s). The neural variants run in 17–23 s/graph but post the largest $\bar{W}_1$ ($\sim$435–465), reflecting both the size of the PDs (max degree $\sim$21, 100+ $H_1$ points) and the difficulty of the topology distribution. CP's $\bar{W}_1 = 0$ is computed over its 33 solved targets; Table 7 compares all methods on the same set of targets.

- **ENZYMES.** The most heterogeneous (real-world). CP and the neural variants are roughly tied on per-graph time (5–7 s); the neural variants achieve $\bar{W}_1 \approx 17$, the lowest TAGG $\bar{W}_1$ relative to PD scale.

### H.6 Qualitative Examples: Detailed Discussion

This appendix expands Figure 3 from Sec. 5.2 with a second test graph and a per-row breakdown of how each method's output deviates from the reference. Both examples are drawn from the ENZYMES test split; node colour encodes degree, which under the degree filtration is also the filtration value of each vertex.

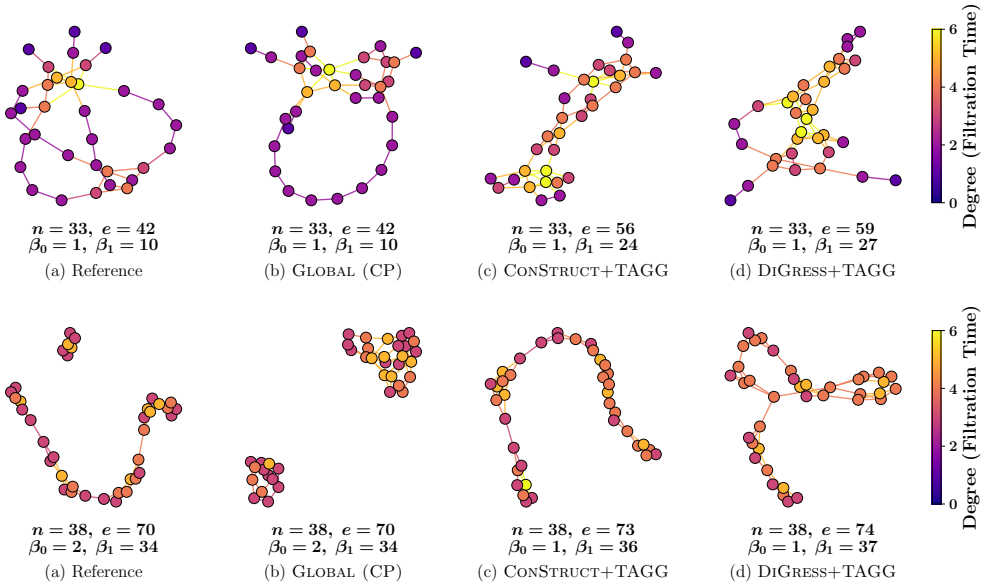



Figure 5: Two qualitative examples on ENZYMES test graphs (`enzymes_87`, top; `enzymes_93`, bottom). Left to right: reference, GLOBAL (CP), CONSTRUCT+TAGG, DIGRESS+TAGG. Node colour encodes degree. CP matches $(n, |E|, \beta_0, \beta_1)$ exactly on both graphs; neural variants drift in edge count and cycle count.

**Top row (`enzymes_87`).** The reference is a moderate-size enzyme contact graph with two connected components and a moderate cycle count. CP reproduces both invariants exactly: the same $\beta_0$, the same $\beta_1$, and the same edge count, with vertices placed into two slots whose union recovers the target PD. The two neural variants both merge the two reference components into a single connected graph ($\beta_0$: $2 \to 1$) and overshoot the target cycle count by 2–3 ($\beta_1$). The merge is the more telling failure mode: it shows that even with the strongest per-target conditioning ($\mu_{G_{\text{test}}}$ rather than the training-set average), the diffusion denoiser cannot reliably preserve component count, because component structure is a global property that no local denoising step on its own can enforce. The excess cycles compound the issue — the PDM regulariser nudges the predicted PD toward the reference but does not commit to it.

**Bottom row (`enzymes_93`).** A larger and denser reference, where the failure modes are amplified. Both neural variants generate 14–17 excess edges and more than double the target $\beta_1$, while $\beta_0$ stays roughly correct. This is the opposite failure pattern from the top row: rather than merging components, the denoiser over-fills the cycle structure within each component. The same structural cause is at work: the PDM loss is a soft penalty on the diagram, not a hard constraint, and when many edge configurations land at similar diffusion-loss values, the gradient-based optimiser settles in a region that is topologically off-target. CP, by contrast, returns a graph that is visibly different from the reference — node positions and edge layout do not match — but every degree-PD invariant is correct. This is the crucial separation our framework makes precise: a strict PD-compliant generator does not aim to recover the reference graph, only some graph in the PD-equivalence class.

# I Complexity and Scaling Analysis

Here we give a complexity and scaling analysis of our local sampler and global encoder when $\Phi = \deg$. Since other filtrations only affects the constraint $\mathcal{C}_1$, the analysis outside of $\mathcal{C}_1$ still applies to them. For the global encoding, we also ran empirical experiments to test its runtimes in practice.

**Notation:** We adopt the following notations in the analysis:

- $n = |\mathcal{P}_0|$ is the vertex count
- $m$ is the edge count
- $K = \max_v \deg(v)$ is the max degree (this can be read off as the max filtration stage)
- $cc_{\max} = \max_k |\{(b, d) \in \mathcal{P}_0 : b \leq k < d\}|$ is the max concurrent alive component count.

## I.1 Local sampler

Each iteration of the sampling selects a vertex-birth or edge-birth event and updates a union-find structure, dominated by a preprocessing sort of the filtration events. This gives a single successful run the overall complexity of $O((n + m) \log(n + m))$ in time and $O(n + m)$ in space. There are no bounds in general on how many runs one needs to try to get a successful outcome (Theorem 3.10), except for when we either have acyclic targets (Corollary 3.4) or stages where cycle-birth demand stays within the running graph's cycle capacity (Proposition 3.3).

## I.2 Global CP encoding

The global encoder is summarized by the following table.

| Constraint | Variables | Constraints |
|---|---|---|
| $\mathcal{C}_1^{\deg}$ | $M_{ij}$: $O(n^2)$ | $O(n)$ |
| $\mathcal{C}_2$ | $x_{kcv}$: $O(K \cdot cc_{\max} \cdot n)$ | $O(Kn)$ |
| $\mathcal{C}_3$ | — | $O(K \cdot cc_{\max} \cdot n^2)$ |
| $\mathcal{C}_4$ | $\pi_{kcij}$: $O(K \cdot cc_{\max} \cdot n^2)$; $\ell_{kcv}, r_{kcv}$: $O(K \cdot cc_{\max} \cdot n)$ | $O(K \cdot cc_{\max} \cdot n^2)$ |
| $\mathcal{C}_5$ | — | $O(K)$ |
| $\mathcal{C}_6$ | $e_{kcb}$: $O(K \cdot cc_{\max})$ | $O(K \cdot cc_{\max})$ |

In total, the encoding uses $O(K \cdot cc_{\max} \cdot n^2)$ variables and constraints. Memory is linear in this quantity (all measured instances fit within commodity RAM). Encoding size is polynomial; solving is NP-hard in general (it subsumes SAT), so no closed-form runtime bound applies. However, modern CP solvers are highly optimized in practice.

Furthermore, we also conducted empirical study of graph structural properties on CP runtime. We ran controlled scaling experiments on synthetic targets, varying one property at a time and running each through the CP solver with a 60s budget. Here we report on some **empirical behaviors** we observed.

1. **Vertices $n$ dominates model size** ($O(n^2)$ edge variables). We tested this on Erdős–Rényi graphs at fixed $p = 0.3$, while varying $n$ (the number of vertices). We obtained on average 0.16s at $n = 10$, 0.87s at $n = 20$, 1.42s at $n = 25$. There is roughly 3-5$\times$ per doubling.
2. **Density helps CP.** We also observed that denser graphs merge components earlier and reduces the quantity $cc_{\max}$. We tested this on Erdős–Rényi graphs at fixed $n = 25$, while varying $p$. We obtained on average 6.83s at $p = 0.2$, 0.90s at $p = 0.5$, 0.60s at $p = 0.7$. Thus, we empirically conclude that density affects cost through $cc_{\max}$.
3. **$cc_{\max}$ is the dominant factor.** We tested this on disjoint unions of $k$ copies of $P_3$, isolating $cc_{\max}$ while keeping $K$ and cycle count fixed. We observed on average 0.03s at $cc_{\max} = 4$, 0.41s at 12, 3.1s at 20, 12.8s at 24. The cost roughly doubles per $+2$ copies.
4. **Cycles add a modest overhead.** We tested this on disjoint unions of copies of $P_3$ against disjoint unions of copies of $C_3$ at matched $cc_{\max}$. At $cc_{\max} = 6$, $6 \times C_3$ (6 cycles) was marginally slower than $6 \times P_3$ (0 cycles), consistent with the extra cycle-birth constraints, though the effect was small relative to the dependence on $cc_{\max}$ itself.

**Remark I.1.** The empirical behaviors above explain the behavior of the **execution time on the SBM dataset** (see Table 2). The SBM graphs have about $100$ vertices on average, and 2-3 communities persisting across many filtration stages before merging. This drives $cc_{\max}$ up over long ranges, hitting both the $n^2$ size term in the encoding and the $cc_{\max}$-multiplier in the slot/parent variable counts simultaneously. The observed SBM timeouts (30-min budget) all have $n \in [144, 164]$ combined with high $cc_{\max}$.

### I.3 A practitioner's guide to choosing the methods

Here we give some heuristics as to which method to choose (local sampler, global CP solver, and neural approximations) in practice.

1. **Local sampler.** This is preferred for sparse targets. Cycle counts grow slowly, so Proposition 3.3 (and Corollary 3.4 for acyclic targets) guarantees single-run termination.
2. **Global CP solver.** This is preferred for denser targets where Theorem 3.10's infeasibility scenarios make local sampling unreliable, provided that the quantities $n$ and $cc_{\max}$ stay tractable. Empirically, we observe runtimes of sub-3s per graph at $n \leq 40$, $cc_{\max} \leq 5$; and sub-30s at $n \leq 60$ or $cc_{\max} \leq 10$.
3. **Neural approximation.** In regimes where both exact methods struggle (e.g., harder SBM instances, as they are not sparse enough for local, and too large in $cc_{\max}$ for CP), neural sampling is the only tractable choice, at the cost of exact PD compliance. As reported in the paper, one-off training cost is substantial, but amortized inference is fast. This pays off when many approximate graphs are needed quickly. Closing the gap in the third regime for our approach likely requires targeted algorithmic advances, or importing established techniques from the constraint-programming literature: symmetry breaking [15, 41], conflict-driven clause learning [31], portfolio solvers [61], problem-specific decomposition [12, 42], none of which our current encoding exploits.

## J Experiments with Other Filtration Schemes

We verify the filtrations — degree, triangle, $k$-clique, clustering, average neighbor degree, closeness, Katz, and PageRank — for both soundness and completeness.

1. For **soundness**, from each generated graph, we recompute $\Phi$ with NetworkX, recompute its PDs, and check it matches the target. Over a suite of structurally diverse graphs (paths, cycles, stars, complete graphs, random trees, Erdős–Rényi, grids, barbells, wheels, Petersen) of size $\leq 30$, every solution realizes the exact target, with no failures.
2. **Completeness** is checked by exhaustive enumeration up to isomorphism at $n \leq 6$: the CP solution set equals the full persistence-equivalence class.

**Remark J.1.** Overall, the solver cost depends on the encoding complexity. For degree, this is linear, whereas triangle and clustering add $O(n^3)$ auxiliaries and are costlier. On COMM-20, the success rate / median per-graph solve time are: degree 100% / 0.16 s, triangle 90% / 0.25 s, clustering 85% / 1.31 s. Scaling these richer filtrations further on remaining datasets and other benchmarks, is a promising direction for future work.

## K Diversity Within a Degree-Based PD-Equivalence Class

Here we test how diverse the graphs in a (**degree-based**) PD-equivalence class are. We first note that diversity is intrinsic to our methods, both of which are sound and complete over the PD-equivalence class. The local sampler produces distinct members through its random choices. The CP-SAT solver enumerates the entire class (all solutions of the encoding), and at scale random seeds return distinct exact members. Thus, in general, we would expect many diverse graphs to occur in a given input PD.

In the experiment, for each reference target we run the CP solver $20\times$ with randomized search, and average pairwise structural distances over targets. The results are displayed below as:

| Dataset | # distinct | clustering | triangle | spectral | planar-rate | PD invariants ($\beta_0$, $\beta_1$, deg, density) |
|---|---|---|---|---|---|---|
| COMM-20 | 20 | 0.064 | 0.190 | 0.059 | 0.087 | 0.000 |
| PLANAR | 18 | 0.020 | 0.212 | 0.020 | 0.000 | 0.000 |
| SBM | 14 | 0.026 | 0.099 | 0.021 | 0.000 | 0.000 |
| ENZYMES | 20 | 0.072 | 0.240 | 0.060 | 0.039 | 0.000 |

To elaborate on the columns of the table:

- **# distinct** is the median number of pairwise non-isomorphic graphs among the 20 solves.
- **clustering** is the mean pairwise absolute difference in average clustering coefficient (in $[0, 1]$).
- **triangle** is the mean pairwise relative difference in triangle count.
- **spectral** is the mean pairwise $\sqrt{n}$-normalized $L_2$ distance between sorted normalized-Laplacian spectra.
- **planar-rate** the fraction of class members that are planar.
- The last column (**PD invariants**) record the mean pairwise absolute difference of the PD-determined invariants ($\beta_0$, $\beta_1$, average degree, density), which is exactly 0.

Note that the spectral column also serves as a surrogate for measuring **graph edit distances**. This is because exact GED is intractable at benchmark sizes ($n$ up to $\sim 123$), so the $\sqrt{n}$-normalized spectral distance is the standard scalable stand-in.

The table confirms that within a degree-PD class, generations diversify in non-topological structure while their PD invariants stay exactly fixed.

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We summarized all contributions in Table 1 and explicitly pointed out where to find them in the paper. The items in Table 1 correspond to items discussed in the abstract and the introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: Yes, we discussed the limitations in Appendix A.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We stated the theoretical results in the main paper and provided their proofs in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the implementation and experimental details in the Appendix H. We also attached the code for the paper in the submission.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide the implementation and experimental details in the Appendix H. We also attached the code for the paper in the submission. Additionally, the code is also hosted online at `https://github.com/Aalto-QuML/ExactCompliance`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Experimental details are explained in Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We conducted experiments over multiple seeds and reported standard deviation, see Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, this is explained in Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have reviewed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes, we included a broader impact statement in Appendix B.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

Justification: The paper does not poses high risk for misuse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cited all the dataset and other methods not proposed in this paper used in the experiments.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We have shared code for all new methods proposed in the paper along with details on the methods.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Yes, we included a declaration of LLM usage in Appendix C.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.