Abstract

Can we use machine learning to compress graph data? The absence of ordering in graphs poses a significant challenge to conventional compression algorithms, limiting their attainable gains as well as their ability to discover relevant patterns. On the other hand, most graph compression approaches rely on domain-dependent handcrafted representations and cannot adapt to different underlying graph distributions. This work aims to establish the necessary principles a lossless graph compression method should follow to approach the entropy storage lower bound. Instead of making rigid assumptions about the graph distribution, we formulate the compressor as a probabilistic model that can be learned from data and generalise to unseen instances. Our “Partition and Code” framework entails three steps: first, a partitioning algorithm decomposes the graph into subgraphs, then these are mapped to the elements of a small dictionary on which we learn a probability distribution, and finally, an entropy encoder translates the representation into bits. All the components (partitioning, dictionary and distribution) are parametric and can be trained with gradient descent. We theoretically compare the compression quality of several graph encodings and prove, under mild conditions, that PnC achieves compression gains that grow either linearly or quadratically with the number of vertices. Empirically, PnC yields significant compression improvements on diverse real-world networks.

1 Introduction

Lossless data compression has been one of the most fundamental and long-standing problems in computer science. It is by now well-understood that the intrinsic limits of compression are governed by the entropy of the underlying data distribution [1]. Crucially, these limits expose an intimate connection between compressibility and machine learning: the better one models the underlying data distribution (from limited observations) the more bits can be saved and vice-versa [2].

The compression of ordered data such as text, images, or video, underpins the modern technology from web protocols to video streaming. However, graph-structured data remain a notable exception. As graph data are becoming more prevalent, it becomes increasingly important to invent practical ways to encode them parsimoniously.

There are three main challenges one faces when attempting to compress graphs:

---

*This work was done while Giorgos Bouritsas was visiting Dr. Andreas Loukas at EPFL, Lausanne. The source code is publicly available at [https://github.com/gbouritsas/PnC](https://github.com/gbouritsas/PnC)
C1. Dealing with graph isomorphism (GI). A key difficulty that distinguishes graphs from conventional data lies in the absence of an inherent ordering of the graph vertices. In order to be able to approach the storage lower bounds, isomorphic graphs should be encoded with the same codeword. However, since the complexity of known algorithms for GI is super-polynomial on the number of vertices [3], a direct use of GI is impractical for graphs consisting of more than a few hundred vertices. Indeed, an examination of the graph compression literature reveals that most progress has been made by optimising a vertex ordering and adapting methods originally invented for vectored data [4–6]. Unfortunately, naively encoding graphs as vectors results in a significant loss in compression.

C2. Evaluating the likelihood. An optimal encoder [1] requires one to accurately estimate and evaluate the probabilities of all the possible outcomes of the underlying domain. When dealing with high-dimensional data, this can be addressed by partitioning the data into parts to obtain a decomposition of the probability distribution: e.g., images can be compressed by modelling the distribution of pixels or patches [10–12], and text by focusing on characters or n-grams [13–19]. However, since graphs do not admit an efficiently computable canonical ordering, it is unclear what decomposition one should employ.

C3. Accounting for the description length of the learned model. The classical learning theory trade-off between model complexity and generalisation is of paramount concern for effective compression. Though in typical deep learning applications one can aim to model the data distribution with an overparametrised neural network (NN) that generalises well, utilising such models to compress information is problematic: since decoding is impossible unless the decoder also receives the learned model (i.e., the NN parameters), overparametrised models are, by definition, suboptimal. This is a pertinent issue for likelihood-based neural approaches as overparametrisation is commonly argued to be a key component of why NNs can be trained [20–22].

The Partition and Code (PnC) framework. Our main contribution is PnC, a framework for learning compression algorithms suitable for encoding graphs sampled from an underlying distribution. In the heart of PnC lie two ideas: (a) Learn to break the problem into parts. Rather than predicting directly the likelihood, we aim to learn how to decompose graphs into non-overlapping subgraphs, see Fig. 1. (b) Identify and code recurring subgraphs when possible. We use a learned dictionary to code subgraphs that appear frequently, whereas rare subgraphs are encoded separately. The dictionary is restricted to contain only a finite number of small recurring subgraphs. This biases the model towards interpretable and well generalising solutions. Both the “partition” and “code” components of PnC are learned directly from the data, by optimising the total description length.

Our framework provides a solution to the three challenges of graph compression. C1: By constraining the dictionary to only contain graphs of size up to a small constant, we can efficiently solve GI. C2: Graph partitioning provides us with the desirable decomposition of the distribution. Using appropriate parametrised probabilistic models, we obtain a closed-form expression for the likelihood that can be later used by any close-to-optimal entropy coder [23–25]. C3: Since we use NNs to decompose the distribution (and not to predict the likelihood), we can rely on overparametrisation without having to relay the NN parameters to the decoder. Also, the complexity of our learned hypothesis (the dictionary) can be computed, and thus optimised, during training.

Theoretical results. Our analysis reveals that PnC can significantly improve upon less sophisticated graph encoders and justifies the usefulness of both the “Partition” and the “Code” component. Specifically, we prove that under mild conditions on the underlying graph distribution, PnC requires in expectation $\Theta(n^2)$ less bits than standard graph encodings, even if the latter are given access to an

---

This follows by a simple counting argument: there are $2^{\binom{n}{2}}$ labelled undirected graphs while the respective number for unlabelled graphs is asymptotically equal to $2^{\binom{n}{2}}/n!$ [7]. Thus, if all graphs of $n$ vertices are equally probable, an encoding that does not consider isomorphism would sacrifice $\log n!$ bits [8,9].
Further, the dictionary induces additional savings of $\Theta(n)$ bits, with the gain being inversely proportional to the entropy of the distribution of the dictionary atoms. Thus, the more repetitive the patterns in the graph distribution are, the larger will be the compression benefits of PnC.

**Practical algorithms.** We instantiate our framework using the following algorithmic modules: (a) a low-parameter learnable estimate of the probability distribution, (b) learning to select the dictionary from a graph universe of finite size, and (c) learning to partition. The latter is a parametric randomised iterative algorithm, the probabilities of which are inferred from a Graph Neural Network (GNN) and optimised with reinforcement learning. Importantly, all algorithms can be jointly trained in order to minimise the total description length in a synergistic manner. We evaluate our framework on diverse real-world graph distributions and showcase compression gains with respect to both conventional and advanced baseline compressors, in observed and unseen data.

2 Related work

**Engineered codecs.** The majority of graph compressors are not probabilistic, but rely on hand-engineered encodings optimised to take advantage of domain-specific properties of e.g., Web-Graphs [4], social networks [6,26,27], and biological networks [28]. A common idea in these approaches is vertex reordering [4–6,26,27,29–31], where the adjacency matrix is permuted in such a way that makes it “compression-friendly” for mainstream compressors of sequences, such as gzip. The algorithms identifying the re-orderings are usually based on heuristics taking advantage of specific network properties, e.g., community structure. Another recurrent idea is to detect or use predefined frequent substructures (e.g., cliques) to represent more efficiently different parts of the graph via grammar rules [32]. These approaches do not attempt to model the underlying graph distribution and thus to approach the storage lower bounds, but strive to find a balance between compression ratios and fast operations on the compressed graphs. Thus, despite their practical importance, they are less relevant to our work. A comprehensive survey can be found in [33].

**Theory-driven approaches.** Several works have contributed to the foundations of the information content and the complexity of graphs [8,9,34–40]. However, few works have attempted to model the underlying graph distribution. Perhaps the most outstanding progress has been made for graphs modelled by the Stochastic Block Model (SBM) [41–48]. Although originally invented for clustering and network analysis purposes, these approaches can be seamlessly used for compression due to their exact likelihood computation. In fact, as we argue in this work, virtually any graph clustering algorithm can be used successfully for compression, by defining codewords corresponding to a community-based random graph model. However, as our experiments confirm, such approaches are less effective at compressing graphs that do not contain clusters.

**Likelihood-based neural approaches.** Any generative model that can provide likelihood estimates in a finite sample space can be used for lossless compression. As a result, a plethora of likelihood-based neural compressors have been recently invented, ranging from autoregressive models for text [18,19,49,50] and images [11,12] to latent variable models [51,55] (paired with bits-back coding [56,57] and Asymmetric Numeral Systems - ANS [25]), normalising flows [58,61] and most recently, diffusion-based generative models [62,63]. However, the vast majority of current graph generators lack the necessary theoretical properties an effective graph compressor should have: they compute probabilities on labelled graphs instead of isomorphism classes by resorting to a heuristic ordering [64,71] (in general this will be suboptimal unless we canonicalise the graph/solve graph isomorphism, while different orderings will have non-zero probabilities, hence we will incur compression losses), and/or do not provide a likelihood [72–76].

An important caveat, which is often ignored in the literature, is that when parametrising the distribution with a neural network, the data cannot be recovered unless the decoder has access to the network itself. Hence it is necessary to account for the NN's description length when evaluating compression gains. However, generative models are usually parameter inefficient, while compressing them (especially during training) is a challenging problem [77]. Note that this is significantly different than compressing neural classifiers, since the capacity to infer the likelihood up to high precision needs to be retained. Therefore, these approaches can have diminishing (or even negative) returns when the dataset size is not large enough. In contrast, by optimising the total description length (equivalent to maximum a-posteriori), we design a compressor that is practical even for small datasets, while the learned dictionary makes our compressor interpretable.
Other related work includes using compression objectives paired with heuristic algorithms for downstream tasks, such as motif finding [78, 79] and graph summarisation [80, 81], lossy compression/coarsening [82, 87], and graph dictionary learning in the context of sparse coding [88, 89].

3 Preliminaries

We use capital letters for sets $A, B$, bold font for vectors $x$ and matrices $X$, and calligraphic font $\mathcal{X}$ for families of sets. We express the information content in bits by using a base-2 logarithm $\log$.

Graphs. Let $G = (V, E)$ be a graph with $n$ vertices $V = \{v_1, \ldots, v_n\}$ and $m$ edges, where $e_{ij} \in E$ whenever vertices $v_i, v_j \in V$ are connected by an edge. For simplicity, we assume the graphs to be undirected, though the same methods apply to directed graphs as well with minimal modifications. Let cut$(A, B) = \{e_{ij} \in E \mid v_i \in A \text{ and } v_j \in B\}$ denote the set of edges with endpoints at two different vertex sets $A, B \subseteq V$.

Information theory. Following the usual terminology in information theory and Minimum Description Length (MDL) theory [2, 90, 91], we assume an observation space $\mathcal{G}$ (in our case a space of possible graphs), and a probability distribution $p$ (sometimes referred to as probabilistic source) producing samples from the observation space. We observe a dataset $\mathcal{G} = \{G_1, G_2, \ldots, G_{|\mathcal{G}|}\}$ of i.i.d. observations drawn from $p$. Note that this setting is in contrast with most works on graph compression [4, 30, 5, 6], where the target is to compress a single large network, such as a social network or a web graph. Let a description method or symbol code $\text{CODE} : \mathcal{G} \to \{0, 1\}^*$ be a mapping from the observation space to a variable-length sequence of binary symbols, the output of which is a codeword.

In the context of graph compression, we are interested in the description length of the code $\text{L}_{\text{CODE}}(G)$, or $L(G)$ for brevity, i.e., the number of bits needed to encode the graph $G$, rather than the code itself, given a single requirement: the code needs to be uniquely decodable, meaning that any concatenation of codes can be uniquely mapped to a sequence of observations. This property can be easily verified using only the description lengths by the well-known Kraft–McMillan inequality (l.h.s. in formula (1)), an important implication of which is that every uniquely decodable code implies a probability distribution $q(G)$ (r.h.s.):

$$\sum_{G \in \mathcal{G}} 2^{-L(G)} \leq 1 \quad \Leftrightarrow \quad q(G) = \frac{2^{-L(G)}}{\sum_{G \in \mathcal{G}} 2^{-L(G)}} \quad (\text{distribution})$$

This equivalence allows us to always define underlying probabilistic models when working directly with description lengths. When the Kraft–McMillan inequality holds with equality, we say that the code is complete; the case of strict inequality means the code is redundant. Moreover, in the case of complete codes, the expected length of the code corresponds to the distribution entropy $H_q[G]$ (plus a constant term when the code is redundant). Thus, compression is synonymous to defining probabilistic models with the lowest possible entropy. An important notion that frequently appears in our theoretical analysis is the binary entropy, i.e., the entropy of a bernoulli variable with probability of success $p$. To distinguish it from the general notion of entropy we will denote it as $H(p) = -p \log p - (1 - p) \log(1 - p)$. It holds that $0 \leq H(p) \leq 1$, where the l.h.s equality is satisfied for $p \in \{0, 1\}$ and the r.h.s. for $p = 1/2$.

Common graph encodings. An important principle that we follow is that whenever we cannot make any assumptions about an underlying distribution, or modelling it is impractical, then the uniform distribution $p_{\text{null}}$ is chosen for encoding. The reason is that uniform distribution is worst-case optimal [90]: for any non-uniform unknown distribution $p$, there always exists a distribution $q$ corresponding to an encoding of which will be on expectation worse than the uniform encoding: $\mathbb{E}_{x \sim p}[-\log q(x)] > \mathbb{E}_{x \sim p}[-\log p_{\text{null}}(x)]$. In the context of graphs, when we cannot make any assumptions or when enumeration is impossible, we will be using a slightly more informative distribution: the Erdős–Rényi (ER) random graph model. This model assigns equal probability to all labelled graphs with $n$ vertices and $m$ edges. Assuming a uniform probability over the possible number of vertices $n$ and number of edges $m$ given $n$, we get:

$$L_{\text{null}}(G) = \log(n_{\text{max}} + 1) + \log \left(\binom{n}{2} + 1\right) + \log \left(\binom{n}{m}\right),$$

where $n_{\text{max}}$ is an upper bound on the number of vertices. The “null” encoding compresses more efficiently graphs that are either very sparse or very dense (low-surprise) as the number of possible graphs with $m$ edges is maximised when $m = n(n - 1)/4$.  

4
4 The Partition & Code (PnC) graph compression framework

Our pipeline consists of three main modules: a partitioning module, a dictionary module, and an entropy encoding module. (a) The partitioning module is responsible for decomposing the graph into disjoint subgraphs and cross-subgraph edges (or cuts). Subgraphs play the role of elementary structures, akin to characters or words in text compression, and pixels or patches in image compression. (b) The dictionary is a small collection of subgraphs (atoms) that are recurrent in the graph distribution. The dictionary module maps the partitioned subgraphs to atoms in the dictionary, allowing us to represent the graph as a collection of atom indices and cuts. (c) Finally, this representation is given as input to the an entropy encoder that translates it into bits.

Decoding the graph in a lossless way involves inverting these three steps: initially the atom indices and cuts are decoded using the same probabilistic model with the encoder, then the atoms are retrieved from the dictionary, and finally all elements are composed back to obtain the original structure. The composition becomes possible by making sure that the cuts are encoded w.r.t. an arbitrarily chosen ordering of each atom’s vertices, hence the decoded graph is guaranteed to be isomorphic to the input, but not necessarily with the same vertex ordering.

4.1 Step 1. Partitioning

The first step of PnC is to employ a parametric partitioning algorithm \( \text{PART}_\theta \) to decompose each graph \( G \) into \( b \) subgraphs of bounded size:

\[
\text{PART}_\theta(G) = (\mathcal{H}, C) \quad \text{and} \quad \mathcal{H} = \{H_1, H_2, \ldots, H_b\},
\]

where \( H_i = \{V_i, E_i\} \) is the \( i \)-th subgraph and \( C = \{V, E_C\} \) is a \( b \)-partite graph containing all cut edges \( E_C = E - \bigcup_i E_i \). Variable \( \theta \) indicates the learnable parameters.

**Is partitioning necessary?** In order to convert probability estimates to codewords, entropy encoders need to be able to compute the probability of every graph in the observation space (or equivalently to have access to the cumulative distribution function (c.d.f.)) - see challenge C2 in Section 1. To achieve this goal, one needs to partition the observation space in a way that permits efficient enumeration of the possible outcomes. Analogously, as we will see in Section 4.3, graph partitioning allows us to decompose the distribution and thus to obtain a closed-form expression for the likelihood. Further, Theorem 1 suggests that partitioning brings a useful inductive bias for graph data providing significant storage gains compared to distribution-agnostic baselines.

4.2 Step 2. Graph dictionary

Rather than naively compressing each subgraph in \( \mathcal{H} \) under a null model, an effective compression algorithm should exploit regularities in the output of \( \text{PART}_\theta \). We propose to utilise a dictionary that stores the most commonly occurring subgraphs. Concretely, we define a dictionary \( D \) to be a collection of connected subgraphs (or atoms) from some universe \( \mathcal{U} \):

\[
D = \{a_1, a_2, \ldots, a_{|D|}\}, \quad \text{where} \quad a_i \in \mathcal{U}.
\]

There are two viable choices for the atom encoding: (a) If the universe is small enough to be efficiently enumerable then we can assume a uniform distribution over \( \mathcal{U} \) which yields the description length \( L(D) = |D| \log |\mathcal{U}| \). Intuitively, this would amount to storing the index of each atom within a list enumerating \( \mathcal{U} \). (b) On the other hand, when \( \mathcal{U} \) is too large to enumerate the atoms can be stored one-by-one based on the null-model encoding given in \( \mathcal{E} \):

\[
L(D) = \sum_{a_i \in D} L_{\text{null}}(a_i).
\]

It is important to note that, to be as effective as possible, the partitioning and the dictionary should be co-designed: \( D \) should capture those subgraphs that are most likely, whereas \( \text{PART}_\theta \) should be biased towards subgraphs with similar structure. The use of a dictionary makes it possible to explicitly account for (and thus optimise) the description length of the learned hypothesis (i.e., equation above), which is an essential component of any compression algorithm (see challenge C3 in Section 1).
4.3 Step 3. Graph encoding

The last step entails compressing \( G \) by encoding the output \((\mathcal{H}, C)\) of \( \text{PART}_\theta \). As discussed in Section 3, a uniquely decodable code implies a probability distribution, i.e., \( q_\phi(\mathcal{H}, C) \) which corresponds to a description length \( L_\phi(\mathcal{H}, C) = -\log q_\phi(\mathcal{H}, C) \), where \( \phi \) denotes the learnable parameters. In the following we explain how \( q_\phi \) is decomposed.

**Subgraphs.** We opt for a dual encoding of subgraphs: one for the subgraphs that belong to the dictionary \( \mathcal{H}_{\text{dict}} \), and one for non-dictionary subgraphs \( \mathcal{H}_{\text{null}} \) that are encoded with the help of a null model as in Eq. (2). This choice has a dual purpose: (a) It allows \( \text{PART}_\theta \) to choose non-dictionary atoms. This is crucial to our approach, since constraining the partitioning to specific isomorphism classes would significantly complicate optimisation. (b) Further, it enables us to maintain a balance between two common structures found in real-world networks—frequent subgraphs (stored in the dictionary), and low-entropy subgraphs as implied by the null model (i.e., very sparse or very dense subgraphs). The distribution is therefore decomposed into the following components:

**Number of subgraphs.** First, we encode the number of dictionary and non-dictionary subgraphs \( (b_{\text{dict}} \text{ and } b_{\text{null}} \text{ respectively}) \) as follows:

\[
q_\phi(b_{\text{dict}}, b_{\text{null}}) = \text{Binomial}(b_{\text{dict}}| b; \phi) q_\phi(b_{\text{null}}) = \left( \binom{b}{b_{\text{dict}}} \right) (1 - \delta_\phi)^{b_{\text{null}}} \delta_\phi^{b_{\text{dict}}} q_\phi(b),
\]

where \( 1 - \delta_\phi = \mathbb{P}[H \in D] \) is the probability of an arbitrary subgraph to belong in the dictionary and \( q_\phi(b) \) is a categorical.

**Dictionary subgraphs.** The dictionary subgraphs are encoded in a permutation invariant way via a multinomial distribution, i.e., we encode the histogram of atoms:

\[
q_\phi(\mathcal{H}_{\text{dict}}| b_{\text{dict}}, D) = \text{Multinomial}(b_1, b_2, \ldots, b_D | b_{\text{dict}}; \phi) = b_{\text{dict}}! \prod_{a \in D} \frac{q_\phi(a)^{b_a}}{b_a!},
\]

where \( b_a = |\{ H \in \mathcal{H}_{\text{dict}} | H \cong a \}| \) and \( \sum_{a \in D} b_a = b_{\text{dict}} \).

**Non-Dictionary subgraphs.** The non-dictionary subgraphs are encoded independently according to the null model:

\[
q_\phi(\mathcal{H}_{\text{null}}| b_{\text{null}}, D) = \prod_{H_i \in \mathcal{H}_{\text{null}}} q_{\text{null}}(H_i).
\]

**Cuts.** We encode the cuts conditioned on the subgraphs, using a non-parametric uninformative null model for multi-partite graphs similar to \([45]\) (see Appendix B.1 for the detailed expression) that prioritises low-entropy cuts. In this way we give more emphasis to the subgraphs and an inductive bias towards distinct clusters in the graph.

Overall, the description length of \((\mathcal{H}, C)\) is \( \text{PART}_\theta(G) \) is given by

\[
L_\phi(\mathcal{H}, C| D) = L_\phi(b_{\text{dict}}, b_{\text{null}}) + L_\phi(\mathcal{H}_{\text{dict}}| b_{\text{dict}}, D) + L_{\text{null}}(\mathcal{H}_{\text{null}}| b_{\text{null}}, D) + L_{\text{null}}(C| \mathcal{H});
\]

and the learnable parameter set is \( \{ \delta_\phi, \{ q_\phi(b) \}_{b=b_{\text{null}}}, \{ q_\phi(a) \}_{a \in D} \} \).

**Remarks about graph isomorphism (GI).** Observe that given a fixed decomposition, our parametrisation is invariant to isomorphism, which is a desirable property since isomorphic graphs will be assigned the same codeword. Note that this is not sufficient to guarantee that all isomorphic graphs will be assigned the same codeword. Since this would imply a solution to GI, it remains an open problem. However, our graph encoding provides desirable tradeoffs between the expressivity of the probabilistic model, its number of parameters, and computational complexity, since it can adapt to different graph distributions using only a few parameters and solving GI only for small graphs.

4.4 Selecting a hypothesis by minimising the total description length

Putting everything together, in order to encode a graph dataset \( \mathcal{G} \) sampled i.i.d. from \( \Theta \) we minimise the total description length

\[
\min_{\theta, D, \phi} \sum_{G \in \mathcal{G}} L_\phi(\text{PART}_\theta(G) | D) + L(D)
\]
with respect to the parameters $\theta$ of the parametric partitioning algorithm, the dictionary $D$, and the parameters $\phi$ of the probabilistic model. Eq. (10) is a typical two-part Minimum Description Length (MDL) objective \cite{29,30}. Using standard MDL terminology, the tuple $(\theta, D, \phi)$ is a point hypotheses and minimising (10) amounts to finding the simplest hypothesis that best describes the data.

5 Theoretical analysis: quadratic and linear gains

The following section performs a comparative analysis of the description length growth rate of various graph compressors. We compare PnC against two strong baselines: (a) The code length $L_{\text{part}}$ induced by a pure partitioning-based graph encoding. Here, a graph is decomposed into subgraphs and cuts, but the distribution of subgraphs is not modelled (i.e., both subgraphs and cuts are encoded with a null model $L_{\text{null}}(G) = L_{\text{null}}(\mathcal{G}) + L_{\text{null}}(\mathcal{C})$). (b) The code length of encodings that do not rely on partitioning but encode each graph as a whole. Importantly, our results hold even for those baselines that encode the isomorphism class of each graph, rather than the graph itself, such as the Erdős-Renyi model for unlabelled graphs of $n$ vertices: $L_{\text{ER-S}}(G) = \log(\Phi_{n,m}) + \log(n^2 + 1)$, where $\Phi_{n,m}$ is the set of all graphs with $n$ vertices and $m$ edges. $L_{\text{ER-S}}(G)$ serves as lower bound to typical encodings such as that of Eq. (2), but can be impractical to implement due to the complexity of GI. The analysis of additional baselines can be found in Appendix A.

Our main theorem shows that, under mild conditions on the underlying graph distribution, the expected description lengths of the compared encodings are totally ordered:

**Theorem 1.** Consider a distribution $p$ over graphs with $n$ vertices and a partitioning algorithm that decomposes a graph into $b$ blocks of $k = O(1)$ vertices. Then it holds that:

$$E_{G \sim p}[L_{\text{PnC}}(G)] \overset{(1b)}{\lesssim} E_{G \sim p}[L_{\text{part}}(G)] \overset{(1a)}{\lesssim} E_{G \sim p}[L_{\text{ER-S}}(G)]$$

under the following conditions:

(1a) $\frac{\log(k^2 + 1)}{k^2} \bar{H}_{m_{ij}} < \bar{H}_{m_i}$, where $\bar{H}_{m_{ij}} = E_{G \sim p}[H\left(\frac{m_{ij}}{k^2}\right)]$ and $\bar{H}_{m_i} = E_{G \sim p}[H\left(\frac{m_i}{n^2}\right)]$ is the expected binary entropy of the cut size $m_{ij}$ between two subgraphs and that of the total number of edges $m_i$, respectively.

(1b) $|D| < (k^2 + 1)^2k^2\bar{H}_{m_i}$, where $|D|$ is the size of the dictionary and $\bar{H}_{m_i} = E_{G \sim p}[H\left(\frac{m_i}{n^2}\right)]$ the expected binary entropy of the number of edges $m_i$ in a subgraph.

The compression gains are:

$$E_{G \sim p}[L_{\text{part}}(G)] \lesssim E_{G \sim p}[L_{\text{ER-S}}(G)] - n^2\left(\bar{H}_m - \frac{\log(k^2 + 1)}{k^2} - \bar{H}_{m_{ij}}\right)$$

and

$$E_{G \sim p}[L_{\text{PnC}}(G)] \lesssim E_{G \sim p}[L_{\text{part}}(G)] - nk(1 - \delta)\left(\bar{H}_{m_i} - \frac{\mathbb{H}(D) - \log(k^2 + 1)}{k^2}\right),$$

where $1 - \delta$ is the probability that a subgraph belongs in the dictionary and $\mathbb{H}(D) = \mathbb{H}_{a \sim q(a)}[a]$ is the entropy of the distribution on dictionary atoms $q(a)$.

Theorem 1 provides insights on the compressibility of certain graph distributions given their structural characteristics. In particular, we can make the following remarks: (a) Condition (1a) can be satisfied even for very small values of $k$ as long as the graphs possess community structure. Perhaps counter-intuitively, when $k = O(1)$ we can satisfy the condition even if the communities have $O(n)$ size by splitting them into smaller subgraphs. This is possible because, in contrast to the majority of graph partitioning objectives that are based on minimum cuts, the compression objective attains its minimum when the cuts have “low entropy”. Since communities that are tightly internally connected have large cuts, $\bar{H}_{m_{ij}}$ and the code length will be kept small. This is a key observation that strongly motivates the use of partitioning for graph compression. (b) Condition (1b) provides an upper bound to the size of the dictionary, which can be easily satisfied for moderately small values of $k$. More importantly, the dependence of the compression gain on the entropy $\mathbb{H}(D)$, reveals that dictionary atoms should be frequent subgraphs in the distribution, confirming our intuition. The bounds also show that, since the probabilities of the atoms are estimated from the data, PnC does not need to make assumptions about the inner structure of the subgraphs and can adapt to general distributions.
In Appendix [A.4] we also provide theoretical evidence on the importance of encoding dictionary subgraphs as isomorphism classes instead of adjacency matrices, which is related to challenge C1 mentioned in the introduction. In particular, Theorem 2 shows that, if isomorphism is not taken into account, the number of bits that will be lost will grow linearly with the number of vertices. All proofs and detailed assumptions can be found in Appendix [A].

6 Optimisation and learning algorithms

We turn our focus to learning algorithms for the optimisation of the MDL objective \((10)\). The following sections explain how each parametric component of PnC is learned.

Subgraph encoding \(\phi\). The graph encoding is parametrised as follows: \(q_\phi(a_i)\) and \(q_\phi(b)\) are parametrised by learnable variables that are converted into categorical distributions over the dictionary atoms and the number of vertices respectively, using a softmax function. Similarly, \(\delta_\phi\) is parametrised by a learnable variable converted to a probability via the sigmoid function.

Dictionary \(D\). Let \(\mathcal{U} = \{a_1, a_2, \ldots a_{|\mathcal{U}|}\}\) be a practically enumerable universe and define \(x = (x_1, x_2, \ldots, x_{|\mathcal{U}|})\) as

\[
x_i = \begin{cases} 
1 & \text{if } a_i \in D \\
0 & \text{otherwise.}
\end{cases}
\]

Thus, \(x_i\) indicates whether \(D\) contains subgraph \(a_i\). Now, optimising w.r.t the dictionary amounts to finding the binary assignments for \(x\) that minimise \((10)\). To circumvent the combinatorial nature of this problem, we apply the continuous relaxation \(\hat{x}_i = \sigma(\psi_i), \forall i \in \{0, 1, \ldots, |\mathcal{U}|\}\), where \(\sigma\) is the sigmoid, \(\psi_i\) are learned continuous variables, and \(\hat{x}_i \in [0, 1]\) a fractional alternative to \(x_i\). Appendix B.3 shows how \((10)\) can be re-written w.r.t. \(x\) and optimised by using the surrogate gradient w.r.t \(\hat{x}_i\).

It is important to note that, in practice, we do not have to introduce indicator variables for the entire universe: Since most subgraphs \(a_i\) will be never encountered in the graph distribution, we build the universe adaptively during training, by progressively adding the different graphs that the partitioning algorithm yields. We also allow the universe to contain subgraphs of size up to \(k = O(1)\), in order to ensure that the isomorphism testing between atoms and subgraphs can be efficiently computed.

Parametric graph partitioning algorithm. Finding the graph partitions that minimise \((10)\) in principle requires searching in the space of partitioning algorithms. Instead, we constrain this space via a differentiable parametrisation that allows us to perform gradient-based optimisation. Currently, learning to partition is an open problem, as to the extent of our knowledge known neural approaches require a fixed number of clusters [93–95] or do not guarantee that the subgraphs are connected [96].

Our Neural Partitioning is a randomised algorithm parametrised with a graph neural network (GNN). When run on a graph, the GNN outputs a random \((H, C)\) together with a corresponding probability \(p_G^{\text{GNN}}(H, C|G)\) and training is performed by estimating the gradients w.r.t. \(\theta\) with REINFORCE [97]. Our algorithm proceeds by iteratively sampling (and removing) subgraphs from the graph until it becomes empty. At each step \(t\) we select a subgraph \(H_t\), by first sampling its vertex count \(k_t\), and subsequently sampling at most \(k_t\) vertices. To guarantee connectivity, we also sample the vertices iteratively and mask-out the probabilities outside the pre-selected vertices’ neighbourhoods. The complexity of the algorithm is \(O(n)\), where \(n\) the number of the vertices of the graph. Please refer to Appendix B.4 for an in-depth explanation of the algorithm and relevant implementation details. We stress that we mainly consider this algorithm as a proof of concept that we ablate against other non-parametric partitioning algorithms. A plethora of solutions can be explored in a parametric setting and we welcome future work in this direction.

MDL objective. Given dataset \(\mathcal{S}\), we train all components by minimising the description length:

\[
L(\mathcal{S}) = L_x(D) + \sum_{G \in \mathcal{S}} E_{(H, C) \sim p_G^{\text{GNN}}(H, C|G)}[L_{\phi, x}(H, C|D)].
\] (14)

Taking the expectation over the GNN output \((H, C) \sim p_G^{\text{GNN}}(H, C|G)\), we calculate the gradients as:

\[
\nabla_\theta L(\mathcal{S}) = \sum_{G \in \mathcal{S}} E[\nabla_{\theta} L_{\phi, x}(H, C|D)], \quad \nabla_x L(\mathcal{S}) = \nabla_x L_x(D) + \sum_{G \in \mathcal{S}} E[\nabla_x L_{\phi, x}(H, C|D)],
\]

and

\[
\nabla_\theta L(\mathcal{S}) = \sum_{G \in \mathcal{S}} E[L_{\phi, x}(H, C|D) \nabla_\theta \ln p_G^{\text{GNN}}(H, C|G)].
\]
Figure 2: PnC + Neural Part. - Most probable graphs in the IMDB-B dataset (left) and the attributed MUTAG dataset (right). Atom types and bond types are represented as vertex and edge attributes.

7 Empirical Results

We evaluate our framework in a variety of datasets: small molecules, proteins and social networks [98–102]. Across all methods, we assume an optimal encoder that attains the entropy lower bound (this is often realistic since modern entropy coders asymptotically approach it) to evaluate the expressive power of each model independently of the encoder. We measure the description length of the data as their negative log-likelihood (NLL) under each probabilistic model, as well as the total description length by adding the cost of the parameters which need to be transmitted to the decoder (see Appendix D for details).

Baselines. We aim to assess representative approaches across the entire spectrum of graph probabilistic models, i.e., from completely uninformative non-parametric distributions to overparametrised neural generative models. We consider the following types of compressors: (a) Null models. We select the uniform model, where all edges are assumed to be sampled independently with probability equal to 0.5, the edge list model, a typical graph representation, and the Erdős-Rényi model (Eq. 2). (b) Partitioning-based. non-parametric methods that aim at grouping vertices in tightly-connected clusters. They can be used for any type of sparse matrix [103] and are based on the assumption that there exists a hidden community structure in the graph. The partitioning algorithms used are SBM fitting [44–47], Louvain [104] and Label Propagation [105] clustering. The encoding we use to encode the clusters corresponds exactly to the SBM assumptions, hence the partitioning-based results are always superior for this approach. (c) Likelihood-based neural compressors. As with any likelihood-based model, graph generative models can be transformed into graph compressors. We evaluate the original GraphRNN [66] and GRAN [68] networks, as well as smaller instantiations that have undergone model compression using the Lottery Ticket Hypothesis algorithm [106].

Results. Tables 1 and 2 report the compression quality of each method measured in terms of the average number of bits required to store each edge in a dataset (bpe). We present four variants of PnC, differing on the type of partitioning algorithm used [105, 104, 47]. We report separately the cost of compressing the data as well as the total cost (including the parameters). Several observations can be made with regards to the baselines:

First off, off-the-shelf likelihood-based neural approaches are poor compressors due to failing to address challenge C3. These models exhibit an unfavorable trade-off between the data and model complexity, often requiring significantly more bpe than the null models. Although model compression techniques can alleviate this tradeoff (especially for larger datasets, e.g., pruned GraphRNN on ZINC), in most of the cases the compression ratios required to outperform PnC are significantly higher than the best that have been reported in the literature (See Table 4 in the Appendix). Perhaps more importantly, it is unclear how to optimise the model description length during training (one of the few exceptions is [107]) and usually model compression might be tedious and is based on heuristics [108–111]. See Appendix C.2 for more details and additional experiments.

In addition, as expected from Theorem (1a), non-parametric clustering algorithms work well when the dataset has a strong community structure, but are not a good choice for more structured datasets. For instance, the best clustering algorithm requires 2.4x more bpe for ZINC than the best PnC. PnC variants achieve the best compression in all datasets considered. This follows from Theorem (1b), since the learned dictionaries are relatively small, and confirms our hypothesis that our framework is sufficiently flexible to account for the particularities of each dataset. As seen, neural partitioning
Table 1: Average bits per edge (bpe) for molecular graph datasets. **First, Second, Third**

<table>
<thead>
<tr>
<th>Method type</th>
<th>Graph type</th>
<th>Dataset name</th>
<th>MUTAG</th>
<th>PTC</th>
<th>ZINC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>data total params</td>
<td>data total params</td>
<td>data total params</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Uniform (raw adjac.)</td>
<td>- 8.44</td>
<td>-</td>
<td>- 17.43</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Edge list</td>
<td>- 7.97</td>
<td>-</td>
<td>- 9.38</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Erdős-Rényi</td>
<td>- 4.78</td>
<td>-</td>
<td>- 5.67</td>
<td>-</td>
</tr>
<tr>
<td>Partitioning (non-parametric)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SBM-Bayes</td>
<td>- 4.62</td>
<td>-</td>
<td>- 5.12</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Louvain</td>
<td>- 4.80</td>
<td>-</td>
<td>- 5.27</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>PropClust</td>
<td>- 4.92</td>
<td>-</td>
<td>- 5.40</td>
<td>-</td>
</tr>
<tr>
<td>Neural (likelihood)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GraphRNN</td>
<td>1.33 3338.21</td>
<td>388K</td>
<td>1.57 1394.59</td>
<td>389K</td>
</tr>
<tr>
<td></td>
<td>GRAN</td>
<td>0.81 12557.75</td>
<td>1460K</td>
<td>2.18 5269.82</td>
<td>1470K</td>
</tr>
<tr>
<td></td>
<td>GraphRNN (pruned)</td>
<td>1.95 12.39</td>
<td>108K</td>
<td>2.16 6.71</td>
<td>110K</td>
</tr>
<tr>
<td></td>
<td>GRAN (pruned)</td>
<td>2.59 24.56</td>
<td>2.23K</td>
<td>4.31 14.00</td>
<td>2.36K</td>
</tr>
<tr>
<td>PrC</td>
<td>PoC + SBM</td>
<td>3.81 4.11</td>
<td>49</td>
<td>4.38 4.79</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>PoC + Louvain</td>
<td>2.20 2.54</td>
<td>47</td>
<td>2.68 3.14</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>PoC + PropClust</td>
<td>2.42 3.03</td>
<td>63</td>
<td>3.38 4.02</td>
<td>178</td>
</tr>
</tbody>
</table>

Table 2: Average bits per edge (bpe) for social and protein graph datasets. **First, Second, Third**

<table>
<thead>
<tr>
<th>Method type</th>
<th>Graph type</th>
<th>Dataset name</th>
<th>PROTEINS</th>
<th>IMDB-B</th>
<th>IMDB-M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>data total params</td>
<td>data total params</td>
<td>data total params</td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>Uniform (raw adjac.)</td>
<td>- 24.71</td>
<td>-</td>
<td>- 2.52</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Edge list</td>
<td>- 10.92</td>
<td>-</td>
<td>- 8.29</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Erdős-Rényi</td>
<td>- 5.46</td>
<td>-</td>
<td>- 1.94</td>
<td>-</td>
</tr>
<tr>
<td>Partitioning (non-parametric)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SBM-Bayes</td>
<td>- 3.98</td>
<td>-</td>
<td>- 0.80</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Louvain</td>
<td>- 3.95</td>
<td>-</td>
<td>- 1.22</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>PropClust</td>
<td>- 4.11</td>
<td>-</td>
<td>- 1.99</td>
<td>-</td>
</tr>
<tr>
<td>Neural (likelihood)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GraphRNN</td>
<td>2.03 156.99</td>
<td>392K</td>
<td>1.03 132.27</td>
<td>395K</td>
</tr>
<tr>
<td></td>
<td>GRAN</td>
<td>1.51 607.96</td>
<td>1545K</td>
<td>0.26 488.88</td>
<td>1473K</td>
</tr>
<tr>
<td></td>
<td>GraphRNN (pruned)</td>
<td>2.63 37.6</td>
<td>2.56K</td>
<td>1.43 1.92</td>
<td>2.38K</td>
</tr>
<tr>
<td></td>
<td>GRAN (pruned)</td>
<td>4.28 5.11</td>
<td>1.78K</td>
<td>0.84 1.75</td>
<td>2.33K</td>
</tr>
<tr>
<td>PrC</td>
<td>PoC + SBM</td>
<td>3.26 3.60</td>
<td>896</td>
<td>0.50 0.54</td>
<td>198</td>
</tr>
<tr>
<td></td>
<td>PoC + Louvain</td>
<td>3.34 3.58</td>
<td>854</td>
<td>0.96 1.02</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>PoC + PropClust</td>
<td>3.42 3.68</td>
<td>866</td>
<td>1.45 1.64</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>PoC + Neural Part.</td>
<td>3.34±0.25</td>
<td>3.51±0.23</td>
<td>717±61</td>
<td>1.00±0.04</td>
</tr>
</tbody>
</table>

performs in every case better, or on par with the combination of PrC with the Louvain algorithm. However, in the social network datasets, the combination of PrC with SBM achieves the best performance. This occurs because these networks fit nicely with the SBM inductive bias (which as a matter of fact is exactly that of low-entropy cuts), and most importantly, due to the fact that the clusters recovered by the SBM are small and repetitive, which makes them ideal for the PrC framework. We also observe that there is room for improvement for the neural partitioning variant, and hypothesise that a more powerful parametrised algorithm can be designed. Since learnable partitioning is still an open problem, we leave this research direction to future work.

Fig. 2 shows the most likely dictionary atoms for the IMDB-B and the MUTAG dataset (also including attributes - Appendix C.3 provides additional experiments). Observe that cliques or near-cliques and typical molecular substructures, such as carbon cycles and junctions are recovered for social networks and molecules respectively. This clearly highlights the connection between compression and pattern mining and provides evidence for potential applications of our framework.

8 Conclusion

This paper marks an important step towards learnable entropy-based graph compression. To the best of our knowledge, our work represents the first attempt to address the basic principles of parametric compressors of unlabelled graphs learned from observations. In addition, we suggest practical instantiations of our framework that can be trained with gradient-based optimisation, accounting for the total description length, hence aiming for the largest possible parsimony. A number of new research questions arise, such as how to design more expressive, albeit parsimonious distribution estimators, how to improve the neural partitioning algorithms, and how to ensure scalability to single large networks that pose a significant challenge w.r.t the memory constraints of GPUs. We hope that our work will inspire further research in this emerging research area.
Acknowledgements and Disclosure of Funding

We would like to thank the anonymous reviewers for their valuable feedback and suggestions to improve our paper. This project was partially funded by the ERC Consolidator Grant No. 724228 - LEMAN. Giorgos Bouritsas is partially supported by a PhD scholarship from the Dpt. of Computing, Imperial College London. Andreas Loukas and Nikolaos Karalias thank the Swiss National Science Foundation for supporting them in the context of the project “Deep Learning for Graph Structured Data”, grant number PZ00P2 179981. Michael Bronstein acknowledges support from Google Faculty awards and the Royal Society Wolfson Research Merit award.

References


