Partition and Code: learning how to compress graphs

Anonymous Author(s)
Affiliation
Address
email

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 elementary structures, then these are mapped to the elements of a small dictionary, and finally, an entropy encoder translates the representation into bits. All three steps are parametric and can be trained with gradient descent. Our algorithms are quantitatively evaluated on diverse real-world networks obtaining significant storage gains with respect to different families of non-parametric and parametric graph compressors.

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:

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

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

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[1].

C2. Evaluating the likelihood. An optimal encoder [1] would require one to evaluate the probabilities of all the possible outcomes of the underlying domain. Hence, when dealing with high-dimensional data the probability distribution has to be decomposed by partitioning the underlying data into parts: e.g., images can be compressed by modelling the distribution of pixels or patches [10, 11], and text by focusing on characters or n-grams [12–18]. 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 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 overparametrized 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), *overparametrized models are, by definition, suboptimal*. This is a pertinent issue for likelihood-based neural approaches as overparametrization is commonly argued to be a key component of why NNs can be trained [19–21].

1.1 Contributions: the Partition and Code (PnC) framework

Learning to compress graphs. 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.

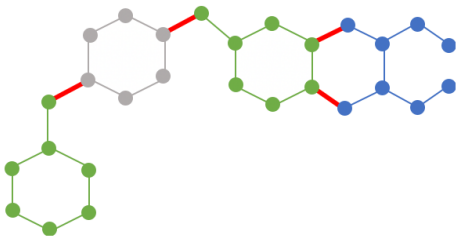

Figure 1: Illustration of the graph decomposition. The subgraph colours correspond to dictionary atoms a_1, a_2 and a_3. Cuts are denoted in red.

Our framework provides a solution to the three challenges of graph compression. C1: PnC induces a decomposition of the probability distribution that is invariant to GI and it does so by comparing only subgraphs of constant size (for which GI can be solved efficiently). C2: Structuring learned information into an efficiently enumerable dictionary allows us to have a closed-form expression for the likelihood and thus to compress data using any close-to-optimal entropy coder [22–24]. C3: Since we use NNs to *decompose* the distribution (and not to predict the likelihood), we can rely on overparametrization 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.

Practical algorithms. We propose a suite of practical algorithms to learn the best hypothesis, including the following: (a) parametrising directly the probability distribution by a small number of learnable parameters, (b) learning to select the dictionary from a predefined 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 algorithms on diverse real-world graph distributions and showcase compression gains with respect to the baselines, both in observed and unseen data.

[1] 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].

2 Related work

Engineered codecs. Although graph compression is a fundamental information-theoretic problem, the majority of the approaches are not probabilistic, but heavily rely on hand-engineered encodings that are optimised to take advantage of certain domain-specific properties, such as WebGraphs [4], social networks [25, 26, 6], and biological networks [27]. A common idea used in these approaches is *vertex reordering* [4, 25, 28, 29, 26, 6, 5, 30], where the adjacency matrix is permuted in such a way that makes it "compression-friendly" for mainstream compressors for sequences such as gzip. The algorithms used to compute 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 [31]. These approaches do not attempt to model the underlying graph distribution and thus to approach the storage lower bounds, but aim at finding 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 [32].

Theory-driven approaches. From a theoretical perspective, several works have contributed to the foundations of the information content and the complexity of graphs [8, 9, 33–38]. However, few works have attempted to model the underlying graph distribution (needed for compression). Perhaps the most outstanding progress has been made for graphs modelled by the Stochastic Block Model (SBM) [39–45]. 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. Finally, graph compression objectives paired with heuristic algorithms have also been used for downstream tasks such as motif finding [46, 47] and graph summarisation [48].

Likelihood-based neural approaches. Given the extraordinary capabilities of deep generative models, there has been an increasing interest for neural approaches to data compression. Usually, a neural network is employed to model the data distribution (e.g., RNNs for text [17, 18, 49, 50], CNNs for images [51, 11, 52]) and then the probability is passed to an entropy coder which translates it into a codeword. Since exact likelihood computation is necessary, the most commonly-used neural networks are VAEs (paired with a technique called bits-back coding [53, 53, 54] and, more recently, normalising flows [55]. Two major difficulties arise when attempting to apply the same rationale to compress graphs: (a) First, current graph generators lack the necessary theoretical properties an effective graph compressor should have: they are not permutation equivariant [56–61] and/or do not provide a likelihood [62–65]. (b) Second, when attempting to use a neural generative model to predict the data likelihood (and thus to design codes), the data cannot be recovered unless the decoder has access to the neural network itself. However, neural generative models are usually parameter inefficient, while compressing them (especially during training) is a challenging problem [66]. 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. Thereferore, 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 (maximum a-posteriori), we design a compressor that is practical even for small datasets, while the learned dictionary makes our compressor interpretable. Importantly, we achieve this without sacrificing the adherence to the theoretical principles of graph probabilistic models.

3 Preliminaries

We use capital letters for sets A, B, bold font for vectors $\mathbf{x}$ and matrices $\mathbf{X}$, 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 $V = \{v_1, \dots, v_n\}$ 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 analysis holds with minimal modifications for directed graphs as well. Let $\mathrm{cut}(A, B) = \{e_{ij} \in E \mid v_i \in A \text{ and } v_j \in B\}$ 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 MDL [2, 67, 68], we assume an *observation space* $\mathfrak{G}$ (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, 29, 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* $\mathsf{CODE} : \mathfrak{G} \to \{0,1\}^*$ 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 $\mathrm{L}_{\mathsf{CODE}}(G)$, or $\mathrm{L}(G)$ 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):

$$\sum_{G \in \mathfrak{G}} 2^{-\,\mathrm{L}(G)} \leq 1 \;\; \text{(code)} \iff q(G) = \frac{2^{-\,\mathrm{L}(G)}}{\sum_{G \in \mathfrak{G}} 2^{-\,\mathrm{L}(G)}} \;\; \text{(distribution)} \tag{1}$$

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 (plus a constant term when the code is redundant). Thus, compression is synonymous to defining probabilistic models that have the lowest possible entropy.

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_{unif} is chosen for encoding. The reason is that uniform distribution is *worst-case optimal* [67]: for any non-uniform unknown distribution p, there always exists a distribution q the corresponding encoding of which will be on expectation worse than the uniform encoding: $\mathbb{E}_{x \sim p}[-\log q(x)] > \mathbb{E}_{x \sim p}[-\log p_{\text{unif}}(x)]$.

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 given n, we get:

$$q_{\text{null}}(G) = \left((n_{\max} + 1) \binom{n}{2} \binom{\binom{n}{2}}{m} \right)^{-1} \iff L_{\text{null}}(G) = \log \left((n_{\max} + 1) \binom{n}{2} \binom{\binom{n}{2}}{m} \right) \tag{2}$$

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 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 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 is a straightforward procedure that 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 PART_θ to decompose each graph G into b subgraphs of bounded size:

$$\mathsf{PART}_\theta(G) = (\mathcal{H}, C) \quad \text{and} \quad \mathcal{H} = \{H_1, H_2, \cdots, H_b\}, \tag{3}$$

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 $E_C = E - \cup_i E_i$. Variable θ indicates the learnable parameters.

Is partitioning necessary? We argue that such decomposition of a graph into smaller manageable parts should be an essential component of any graph compression method. Graph compression boils down to knowing how likely each graph is under the data distribution. However, there is an important caveat: simply knowing the likelihood of an event is not enough! The reason for that is that entropy encoders [22–24] need to be able to compute the probability of *any* possible outcome of the distribution to translate probabilities into codewords (see challenge C2 in Section 1). As we will see in Section 4.3, graph partitioning allows us to decompose the learned distribution and thus to obtain a closed-form expression for the likelihood.

4.2 Step 2. Graph dictionary

Rather than naively compressing each subgraph in (3) under a null model, an effective compression algorithm should exploit regularities in the output of PART_θ. 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 $\mathfrak{U}$:

$$D = \{a_1, a_2, \cdots, a_{|D|}\}, \quad \text{where} \quad a_i \in \mathfrak{U}. \tag{4}$$

From a probabilistic perceptive, the dictionary reflects our prior knowledge with regards to the graph distribution at hand. Opting to prioritise small dictionaries, we encode each atom independently. 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 $\mathfrak{U}$ which yields the description length $\mathrm{L}(D) = |D| \log |\mathfrak{U}|$. Intuitively, this would amount to storing the index of each atom within a list enumerating $\mathfrak{U}$. (b) On the other hand, when $\mathfrak{U}$ is too large to enumerate the atoms can be stored one-by-one based on the null-model encoding given in (2):

$$\mathrm{L}(D) = \sum_{a_i \in D} \mathrm{L}_{\text{null}}(a_i). \tag{5}$$

It is important to note that, to be as effective as possible the partitioning and dictionary should be co-designed: D should capture those subgraphs that are most likely, whereas PART_θ 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). Accounting for both the model and the data is an essential component of any compression algorithm (see challenge C1 in Section 1).

4.3 Step 3. Graph encoding

The last step entails compressing G by encoding the output $(\mathcal{H}, C)$ of PART_θ with the dictionary. As discussed in Section 3, a uniquely decodable encoding implies a probability distribution. In general, we can define a probability distribution on graph decompositions parametrised by ϕ as $q_\phi(G) = q_\phi(\mathcal{H}, C)$ which corresponds to a description length $\mathrm{L}_\phi(\mathcal{H}, C) = -\log q_\phi(\mathcal{H}, C)$. In the following we explain how q_ϕ is decomposed.

An essential property that $q_\phi(G)$ should have is that it needs to be invariant to isomorphism, i.e., that a graph should have the same probability (and hence be assigned the same codeword), regardless of the ordering of its vertices. In our case, this means that decompositions should be assigned the same probability, regardless of the order of the subgraphs, or formally q_ϕ should be *jointly exchangeable* [69] w.r.t. $\mathcal{H}$ and C.

Given that the space of all possible $(\mathcal{H}, C)$ decompositions is enormous, we are unlikely to encounter enough observations to optimise the parameters of an arbitrary jointly exchangeable distribution, but

most importantly the outcomes will still be impossible to enumerate. Thus, we choose a parametrisation that is sufficiently flexible to fit the observed data, without compromising enumeration.

Subgraphs. We encode each H_i independently by verifying if it is isomorphic to a dictionary atom:[2]

$$q_\phi(H_i|D) \propto \begin{cases} q_\phi(a_i) & \text{if } \exists\, a_i \in D \text{ such that } H_i = a_i \\ q_{\text{null}}(H_i) & \text{otherwise.} \end{cases} \tag{6}$$

This choice has a dual purpose: (a) It allows PART_θ to choose non-dictionary atoms, which is central 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 important inductive biases of real-world networks—*frequent subgraphs* (stored in the dictionary), and low-entropy structure as implied by the null model (i.e., very sparse or very dense subgraphs).

Cuts. We encode the cuts conditioned on the subgraphs, using a non-parametric uninformative null model for multi-partite graphs similar to [42] (see supplementary material for the detailed expression) that prioritises small 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 $(\mathcal{H}, C) = \mathsf{PART}_\theta(G)$ is given by

$$\mathrm{L}_\phi((\mathcal{H}, C)|D) = \sum_{i=1}^{b} \mathrm{L}_\phi(H_i|D) + \mathrm{L}_{\text{null}}(C|\mathcal{H}) + \mathrm{L}_\phi(b) = -\log\left(q_{\text{null}}(C|\mathcal{H})\, q_\phi(b) \prod_{i=1}^{b} q_\phi(H_i|D)\right),$$

where $\mathrm{L}_\phi(b) = -\log q_\phi(b)$ encodes the number of blocks in the partition.

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 $\mathfrak{G}$ we minimise the total description length

$$\min_{\theta, D, \phi} \sum_{G \in \mathcal{G}} \mathrm{L}_\phi(\mathsf{PART}_\theta(G) \,|\, D) + \mathrm{L}(D) \tag{7}$$

with respect to the parameters θ of the parametric partitioning algorithm, the dictionary D, and the parameters ϕ controlling the likelihood of each atom from D in a given G.

Minimising the above equation is a typical two-part Minimum Description Length (MDL) objective [70, 67]. Using standard MDL terminology, the tuple (θ, D, ϕ) is a *point hypotheses* and minimising (8) amounts to finding the simplest hypothesis that can best describe the data. Alternatively, this can be seen as maximising the posterior distribution of the parameters.

4.5 Special cases of note

A pertinent question is whether one can determine the optimal way to partition a graph when minimising (8). Though a rigorous statement is beyond our current understanding, in the following we discuss two special cases:

(a) Small predefined universe. When the subgraphs are chosen from a small and predefined $\mathfrak{U}$, one may attempt to identify all the possible atom appearances in G by repeatedly calling a subgraph isomorphism subroutine. The minimisation of (8) then simplifies to that of selecting a subset of subgraphs that have no common edges (as per the definition in Section 4.1). The latter problem can be cast as a discrete optimisation problem under an independent set constraint (by building an auxiliary graph the vertices of which are candidate subgraphs and two vertices are connected by an edge when two subgraphs overlap, and looking for an independent set that minimises the description length).

(b) Unconstrained universe. When $\mathfrak{U}$ contains all possible graphs, the problem can be seen as a special graph partitioning problem. However, contrary to traditional clustering algorithms [71–74], our objective is not necessary optimised by finding small cuts between clusters, but also depends on the selected probabilistic model.

[2] In general, an additional bernoulli variable can be used to indicate if the subgraph belongs to the dictionary. When its probability is equal to 0.5, this amounts to an extra bit of information.

Since most independent set and partitioning problems are NP-hard, we suspect that similar arguments can be put forward for (8). Thus, in the following section, we will propose learnable alternatives that can provide fast solutions without the need to be adapted to unseen data. Nevertheless, both learned and algorithmic approaches will be evaluated in the experiments section.

5 Optimisation and learning algorithms

We turn our focus to learning algorithms for the optimisation of the MDL objective (8). The following sections explain how each parametric component of PnC is learned.

Subgraph encoding ϕ. 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.

Dictionary D. Let $\mathfrak{U}$ be a practically enumerable universe and define $\mathbf{x} = (x_1, x_2, \ldots, x_{|\mathfrak{U}|})$ as

$$x_i = \begin{cases} 1 & \text{if } \mathfrak{U}_i \in D \\ 0 & \text{otherwise.} \end{cases}$$

Thus, x_i indicates whether D contains subgraph $\mathfrak{U}_i$. Now, optimising w.r.t the dictionary amounts to finding the binary assignments for $\mathbf{x}$ that minimise (8). To circumvent the combinatorial nature of this problem, we apply the continuous relaxation $\hat{x}_i = \sigma(\psi_i), \forall i \in \{0, 1, \ldots, |\mathfrak{U}|\}$, where σ is the sigmoid function, ψ_i are learned continuous variables, and $\hat{x_i} \in [0, 1]$ is a fractional alternative to x_i. The supplementary material shows how (8) can be re-written w.r.t. $\mathbf{x}$ 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 $\mathfrak{U}_i$ 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 PART_θ. Finding the graph partitions that minimise (8) would, in principle, require searching in the space of all possible partitioning algorithms. Instead, we constrain this space via a differentiable parametrisation that allows us to perform traditional gradient-based optimisation. Currently, learning to partition graphs is an open problem, as to the extend of our knowledge known neural approaches require a fixed number of clusters [75–77] or do not guarantee that the subgraphs are connected [78]. We propose a neural algorithm that partitions a graphs by iteratively sampling connected subgraphs. The algorithm is parametrised by a graph neural network (GNN). When run on a graph, the GNN outputs a random $(\mathcal{H}, C)$ together with a corresponding probability $p_\theta^{\text{GNN}}(\mathcal{H}, C|G)$ indicating its confidence. We train the GNN by minimising (8) using the REINFORCE [79] algorithm. Please refer to the supplementary material for a more detailed explanation of the algorithm and the 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 different solutions can be explored in a parametric setting and we welcome future work in this direction.

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

$$L(\mathcal{G}) = \mathrm{L}_{\mathbf{x}}(D) + \sum_{G \in \mathcal{G}} \mathbb{E}_{(\mathcal{H}, C) \sim p_\theta^{\text{GNN}}(\mathcal{H}, C|G)} [\mathrm{L}_{\phi, \mathbf{x}}(\mathcal{H}, C|D)]. \tag{8}$$

The gradients w.r.t ϕ, θ, and $\mathbf{x}$ are, respectively, calculated as follows:

$$\nabla_\phi L(\mathcal{G}) = \sum_{G \in \mathcal{G}} \mathbb{E}[\nabla_\phi \, \mathrm{L}_{\phi, \mathbf{x}}(\mathcal{H}, C)|D)] \tag{9}$$

and

$$\nabla_\theta L(\mathcal{G}) = \sum_{G \in \mathcal{G}} \mathbb{E}[\mathrm{L}_{\phi, \mathbf{x}}(\mathcal{H}, C|D) \nabla_\theta \ln p_\theta^{\text{GNN}}(\mathcal{H}, C|G)] \tag{10}$$

$$\nabla_{\hat{\mathbf{x}}} L(\mathcal{G}) = \nabla_{\hat{\mathbf{x}}} \, \mathrm{L}_{\hat{\mathbf{x}}}(D) + \sum_{G \in \mathcal{G}} \mathbb{E}[\nabla_{\hat{\mathbf{x}}} \, \mathrm{L}_{\phi, \hat{\mathbf{x}}}(\mathcal{H}, C|D)], \tag{11}$$

with the expectation taken over the GNN output $(\mathcal{H}, C) \sim p_\theta^{\text{GNN}}(\mathcal{H}, C|G)$.

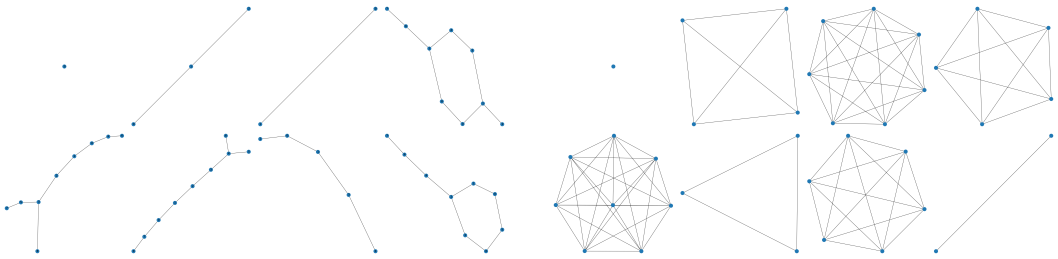

Figure 2: Left: Most probable graphs in the ZINC dataset (PnC + Neural Part), Right: Most probable graphs in the IMDB-M dataset (PnC + SBM).

6 Empirical Results

We evaluate our framework in a variety of datasets from different distributions: small molecules, protein and social networks [80–84].

Experimental setup. Across all methods, we assume an optimal encoder that attains the entropy lower bound, i.e., translates probabilities to codewords computing the negative log likelihood. This assumption is made in order to evaluate the expressive power of each model independently of the encoder. Since modern entropy coders do approach the lower bound, this assumption is not unrealistic. We measure the description length of the data as their negative log-likelihood under each probabilistic model, and most importantly we evaluate the total description length by adding the cost of the parameters which need to be transmitted to the decoder (we assume a 16-bit/parameter representation).

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.* The graphs are encoded as if they originated from a non-parametric null model. 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 *Erdos-Renyi* model, as described in section 3. Note that these models are heavily redundant for large graphs, since they are completely oblivious to graph isomorphism. (b) *Clustering-based.* non-parametric methods that aim at grouping vertices in tightly-connected clusters. They can be used for any type of sparse matrix [85] and are based on the assumption that there exists a hidden community structure in the graph. They can be seen as vertex-reordering methods with the additional information of the cluster sizes. The partioning algorithms used for graph clustering are *SBM-fitting* [41–44], the Louvain algorithm [73] and clustering by Label Propagation [86]. The encoding we use to encode the clusters corresponds exactly to the SBM assumptions, hence the clustering-based results are always superior for this approach. (c) *Likelihood-based neural compressors.* As with any likelihood-based generative model, graph generative models can be transformed into graph compressors. We evaluate graphRNN [57] and GRAN [58], by measuring the negative log-likelihood on the data after training, together with the description length of the network parameters

Results. Tables 1 and 2 report how well each method can compress each dataset. We present four variants of PnC, differing on the type of partitioning algorithm utilised [86, 73, 44]. The compression quality is measured in terms of the average number of bits required to store each each in a dataset (bpe): the fewer bpe are required, the better the compression quality. The tables report separately the cost of compressing the data as well as the total cost (including parameters).

Several observations can be made with regards to the baselines: *First off, likelihood-based neural approaches are poor compressors due to failing to address challenge C3.* As seen, these models exhibit an unfavorable trade-off between the data and model complexity, often requiring significantly more bpe than the null models. *In addition, non-parametric clustering algorithms work well when the datasets has a strong community structure, but are not a good choice for more structured datasets.* For instance, the best clustering algorithm requires $2.1\times$ more bpe for ZINC than the best baseline.

PnC variants achieve the best compression in all datasets considered. This confirms our hypothesis that our framework is sufficiently flexible to account for the particularities of each dataset. As seen, neural partitioning performs in every case on par with the combination of PnC with the Louvain algorithm (or slightly better). Fig. 2 (left) shows the most likely atoms in the dictionary learned for the

Table 1: Average bits per edge (bpe) for molecular graph datasets

Method type	Graph type	Small Molecules								
	Dataset name	MUTAG			PTC			ZINC		
		data	total	params	data	total	params	data	total	params
Null	Uniform (raw adjac.)	-	8.44	-	-	17.43	-	-	10.90	-
	Edge list	-	7.97	-	-	9.38	-	-	8.60	-
	Erdos-Renyi	-	4.78	-	-	5.67	-	-	5.15	-
Clustering (non-parametric)	SBM-Bayes	-	4.62	-	-	5.12	-	-	4.75	-
	Louvain	-	4.80	-	-	5.27	-	-	4.77	-
	PropClust	-	4.92		-	5.40		-	4.85	
Neural (likelihood)	GraphRNN	1.33	1669.77	388K	1.57	698.08	389K	1.62	22.39	388K
	GRAN	0.81	6279.28	1460K	2.18	2636	1470K	1.30	79.50	1461K
PnC	PnC + SBM	3.81	4.16	53	4.55	4.94	155	3.48	3.60	613
	PnC + Louvain	2.41	**2.69**	45	2.81	**3.31**	166	2.14	**2.17**	196
	PnC + PropClust	2.53	**3.14**	63	3.49	**4.14**	178	2.36	**2.53**	725
	PnC + Neural Part.	2.38	**2.65**	45	2.73	**3.01**	147	2.19	**2.22**	242

Table 2: Average bits per edge (bpe) for social and protein graph datasets

Method type	Graph type	Biology			Social Networks					
	Dataset name	PROTEINS			IMDB-B			IMDB-M		
		data	total	params	data	total	params	data	total	params
Null (non-parametric)	Uniform (raw adjac.)	-	24.71	-	-	2.52	-	-	1.83	-
	Edge list	-	10.92		-	8.29	-	-	7.74	-
	Erdos-Renyi	-	5.46	-	-	1.94	-	-	1.32	-
Clustering (non-parametric)	SBM-Bayes	-	3.98	-	-	0.80	-	-	0.60	-
	Louvain	-	3.95	-	-	1.22	-	-	0.88	-
	PropClust	-	4.11		-	1.99	-	-	1.37	-
Neural (likelihood)	GraphRNN	2.03	79.51	392K	1.03	66.65	395K	0.72	64.28	392K
	GRAN	1.51	304.735	1545K	0.26	244.57	1473K	0.17	237.65	1467K
PnC	PnC + SBM	3.41	**3.67**	871	0.53	**0.58**	205	0.37	**0.40**	147
	PnC + Louvain	3.35	**3.62**	854	0.96	**1.01**	202	0.66	**0.70**	141
	PnC + PropClust	3.46	3.74	866	1.45	1.64	241	0.93	1.04	178
	PnC + Neural Part.	3.14	**3.32**	747	1.05	**1.08**	180	0.73	**0.78**	162

ZINC dataset. We observe that typical molecular substructures, such as carbon cycles and junctions can be recovered. However, in the social network datasets the combination of PnC with SBM achieves the best performance. This occurs because these networks fit nicely with the SBM inductive bias, and most importantly, due to the fact that the clusters recovered by the SBM are small and repetitive (see Figure 2), which makes them ideal for the PnC 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. We refer to the Supplementary Material for additional visualisations taking into account also node and edge attributes.

7 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 for isomorphism-invariant parametric graph compressors learned from observations. In addition, we suggest practical instantiations of our framework that can be trained with off-the-shelf gradient optimisers 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, jointly exchangeable distributions, how to improve neural partitioning algorithms that can be incorporated in neural architectures, and how to ensure scalability to single large networks that pose a significant challenge w.r.t the memory constraints of GPUs. The promising experimental results make us hope that our work will inspire further research in this emerging research area.

References

[1] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[2] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[3] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

[4] Paolo Boldi Sebastiano Vigna and P Boldi. The webgraph framework i: compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, pages 595–602, 2004.

[5] Yongsub Lim, U Kang, and Christos Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3077–3089, 2014.

[6] Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1535–1544, 2016.

[7] Frank Harary and Edgar M Palmer. *Graphical enumeration*. Elsevier, 2014.

[8] Moni Naor. Succinct representation of general unlabeled graphs. *Discrete Applied Mathematics*, 28(3):303–307, 1990.

[9] Yongwook Choi and Wojciech Szpankowski. Compression of graphical structures: Fundamental limits, algorithms, and experiments. *IEEE Transactions on Information Theory*, 58(2):620–638, 2012.

[10] Jon Sneyers and Pieter Wuille. Flif: Free lossless image format based on maniac compression. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 66–70. IEEE, 2016.

[11] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10629–10638, 2019.

[12] Terry A Welch. A technique for high-performance data compression. *Computer*, 17(06):8–19, 1984.

[13] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[14] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.

[15] Timothy Bell, Ian H Witten, and John G Cleary. Modeling for text compression. *ACM Computing Surveys (CSUR)*, 21(4):557–591, 1989.

[16] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer, 1994.

[17] Jürgen Schmidhuber and Stefan Heil. Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1):142–146, 1996.

[18] Matthew V Mahoney. Fast text compression with neural networks. In *FLAIRS conference*, pages 230–234, 2000.

[19] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2018.

[20] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pages 244–253. PMLR, 2018.

[21] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

[22] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[23] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[24] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

[25] Alberto Apostolico and Guido Drovandi. Graph compression by bfs. *Algorithms*, 2(3):1031–1044, 2009.

[26] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*, pages 587–596, 2011.

[27] Leonid Peshkin. Structure induction by lossless graph compression. In *2007 Data Compression Conference (DCC'07)*, pages 53–62. IEEE, 2007.

[28] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228, 2009.

[29] Francisco Claude and Gonzalo Navarro. Fast and compact web graph representations. *ACM Transactions on the Web (TWEB)*, 4(4):1–31, 2010.

[30] Ryan A Rossi and Rong Zhou. Graphzip: a clique-based sparse graph compression method. *Journal of Big Data*, 5(1):1–14, 2018.

[31] Sebastian Maneth and Fabian Peternek. Compressing graphs by grammars. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 109–120. IEEE, 2016.

[32] Maciej Besta and Torsten Hoefler. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *arXiv preprint arXiv:1806.01799*, 2018.

[33] Ernesto Trucco. On the information content of graphs: compound symbols; different states for each point. *The bulletin of mathematical biophysics*, 18(3):237–253, 1956.

[34] János Körner. Coding of an information source having ambiguous alphabet and the entropy of graphs. In *6th Prague conference on information theory*, pages 411–425, 1973.

[35] György Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294, 1984.

[36] Matthias Dehmer. Information-theoretic concepts for the analysis of complex networks. *Applied Artificial Intelligence*, 22(7-8):684–706, 2008.

[37] Matthias Dehmer, Nicola Barbarini, Kurt Varmuza, and Armin Graber. A large scale analysis of information-theoretic network complexity measures using chemical structures. *PLoS One*, 4(12):e8057, 2009.

[38] Abbe Mowshowitz and Matthias Dehmer. Entropy and the complexity of graphs revisited. *Entropy*, 14(3):559–570, 2012.

[39] Martin Rosvall and Carl T Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[40] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[41] Tiago P Peixoto. Entropy of stochastic blockmodel ensembles. *Physical Review E*, 85(5):056122, 2012.

[42] Tiago P Peixoto. Parsimonious module inference in large networks. *Physical review letters*, 110(14):148701, 2013.

[43] Tiago P Peixoto. Nonparametric bayesian inference of the microcanonical stochastic block model. *Physical Review E*, 95(1):012317, 2017.

[44] Tiago P Peixoto. Bayesian stochastic blockmodeling. *Advances in network clustering and blockmodeling*, pages 289–332, 2019.

[45] Clement Lee and Darren J Wilkinson. A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4(1):1–50, 2019.

[46] Nikhil S Ketkar, Lawrence B Holder, and Diane J Cook. Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 71–76, 2005.

[47] Peter Bloem and Steven de Rooij. Large-scale network motif analysis using compression. *Data Mining and Knowledge Discovery*, 34(5):1421–1453, 2020.

[48] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99. SIAM, 2014.

[49] David Cox. Syntactically informed text compression with recurrent neural networks. *arXiv preprint arXiv:1608.02893*, 2016.

[50] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Deepzip: Lossless data compression using recurrent neural networks. *arXiv preprint arXiv:1811.08162*, 2018.

[51] Emiel Hoogeboom, Jorn W. T. Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12134–12144, 2019.

[52] Fabian Mentzer, Luc Van Gool, and Michael Tschannen. Learning better lossless compression using lossy compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6638–6647, 2020.

[53] James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[54] Friso Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning*, pages 3408–3417. PMLR, 2019.

[55] Yangjun Ruan, Karen Ullrich, Daniel Severo, James Townsend, Ashish Khisti, Arnaud Doucet, Alireza Makhzani, and Chris J Maddison. Improving lossless compression rates via monte carlo bits-back coding. *arXiv preprint arXiv:2102.11086*, 2021.

[56] Thomas N Kipf and Max Welling. Variational graph auto-encoders. In *NIPS Workshop onBayesian Deep Learning*, 2016.

[57] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018.

[58] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pages 4255–4265, 2019.

[59] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[60] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[61] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Constrained graph variational autoencoders for molecule design. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7806–7815, 2018.

[62] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[63] Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. Conditional structure generation through graph variational generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 1340–1351, 2019.

[64] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816*, 2018.

[65] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. volume 108 of *Proceedings of Machine Learning Research*, pages 4474–4484, Online, 26–28 Aug 2020. PMLR.

[66] Thomas Bird, Friso H. Kingma, and David Barber. Reducing the computational cost of deep generative models with binary. In *International Conference on Learning Representations*, 2021.

[67] Peter D Grünwald and Abhijit Grunwald. *The minimum description length principle*. MIT press, 2007.

[68] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[69] Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetries and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61, 2020.

[70] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[71] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14:849–856, 2001.

[72] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[73] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[74] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30. Citeseer, 1993.

[75] Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. End to end learning and optimization on graphs. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4674–4685, 2019.

[76] Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. GAP: generalizable approximate graph partitioning framework. *CoRR*, abs/1903.00614, 2019.

[77] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. 2020.

[78] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *NeurIPS 2020 34th Conference on Neural Information Processing Systems*, number POST_TALK, 2020.

[79] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[80] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[81] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

[82] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[83] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[84] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[85] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S Modha, and Christos Faloutsos. Fully automatic cross-associations. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 79–88, 2004.

[86] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The standard deviations are included in the supplementary material. For the vast majority of the methods compared, we observe small variations across different runs.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [N/A]

   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]