NeuroCUT: A Neural Approach for Robust Graph Partitioning

Anonymous Author(s)*

ABSTRACT

Graph partitioning aims to divide a graph into k disjoint subsets while optimizing a specific partitioning objective. The majority of formulations related to graph partitioning exhibit NP-hardness due to their combinatorial nature. Conventional methods, like approximation algorithms or heuristics, are designed for distinct partitioning objectives and fail to achieve generalization across other important partitioning objectives. Recently machine learning-based methods have been developed that learn directly from data. Further, these methods have a distinct advantage of utilizing node features that carry additional information. However, these methods assume differentiability of target partitioning objective functions and cannot generalize for an unknown number of partitions, i.e., they assume the number of partitions is provided in advance. In this study, we develop NeuroCUT with two key innovations over previous methodologies. First, by leveraging a reinforcement learning-based framework over node representations derived from a graph neural network and positional features, NeuroCUT can accommodate *any* optimization objective, even those with non-differentiable functions. Second, we decouple the parameter space and the partition count making NeuroCUT *inductive* to any unseen number of partition, which is provided at query time. Through empirical evaluation, we demonstrate that NeuroCUT excels in identifying high-quality partitions, showcases strong generalization across a wide spectrum of partitioning objectives, and exhibits strong generalization to unseen partition count.

KEYWORDS

Graph Partitioning, Min Cut, Robustness, Versatile objectives, Inductive learning, GNN.

ACM Reference Format:

Anonymous Author(s). 2024. NeuroCUT: A Neural Approach for Robust Graph Partitioning. In *30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/XX

1 INTRODUCTION

Graph partitioning is a fundamental problem in network analysis with numerous real-world applications in various domains such as system design in online social networks [31], dynamic ride-sharing in transportation systems [36], VLSI design [16], and preventing cascading failure in power grids [23]. The goal of graph partitioning is to divide a given graph into disjoint subsets where nodes within each subset exhibit strong internal connections while having limited connections with nodes in other subsets. Generally speaking, the aim is to find somewhat balanced partitions while minimizing the number of edges across partitions.

1.1 Related Works

Several graph partitioning formulations have been studied in the literature, mostly in the form of discrete optimization [2, 7, 17–19]. The majority of the formulations are NP-hard and thus the proposed solutions are either heuristics or algorithms with approximate solutions [17]. Among these, two widely used methods are Spectral Clustering [29] and hMETIS [17]. Spectral Clustering partitions a graph into clusters based on the eigenvectors of a similarity matrix derived from the graph. hMETIS is a hypergraph partitioning method that divides a graph into clusters by maximizing intra-cluster similarity while minimizing inter-cluster similarity. However, such techniques are confined to specific objective functions and are unable to leverage the available node features in the graph. Recently, there have been attempts to solve graph partitioning problem via neural approaches. The neural approaches have a distinct advantage that they can utilize node features. Node features supply additional information and provide contextual insights that may improve the accuracy of graph partitioning. For instance, [39] has recognized the importance of incorporating node attributes in graph clustering where nodes are partitioned into disjoint groups. Note that there are existing neural approaches [14, 20, 21, 26] to solve other NP-hard graph combinatorial problems (e.g., minimum vertex cover). However these methods are not generic enough to solve graph partitioning. It should be noted that although S2VDQN [20] learns to solve the Maxcut problem, it is designed for the specific case of bi-partitioning. This hinders its applicability to the target problem setup of k-way partitioning.

In this paper, we build a single framework to solve several graph partitioning problems. One of the most relevant to our work is the method DMoN by Tsitsulin et al. [37]. This method designs a neural architecture for cluster assignments and use a modularity-based objective function for optimizing these assignments. Another method, that is relevant to our work is GAP, which is an unsupervised learning method to solve the balanced graph partitioning problem [28]. It proposes a differentiable loss function for partitioning based on a continuous relaxation of the normalized cut formulation. Deep-MinCut being an unsupervised approach learns both node embeddings and the community structures simultaneously where the objective is to minimize the mincut loss [8]. Another method solves the multicut problem where the number of partitions is *not* an input to the problem [15]. The idea is to construct a reformulation of the multicut ILP constraints to a polynomial program as a loss function. However, the problem formulation is different than the generic normalized cut or mincut problem. Finally, [10] solves the normalized cut problem only for the case where the number of partitions is exactly two. Nevertheless, these neural approaches for the graph partitioning problem often do not use node features and

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN XX…$XX
https://doi.org/XX

only limited to a distinct partitioning objective. Here, we point out notable drawbacks that we address in our framework.

- **Non-inductivity to partition count:** The number of partitions required to segment a graph is an input parameter. Hence, it is important for a learned model to generalize to any partition count without retraining. In -way graph partitioning, a model demonstrates inductivity to the number of partitions when it can infer on varying partition numbers without specific training for each. Existing neural approaches are non-inductive to the number of partitions, i.e they can only infer on number of partitions on which they are trained. Additionally, it is worth noting that the optimal number of partitions is often unknown beforehand. This capability is crucial in practical applications like chip design, where graph partitioning optimizes logic cell placement by dividing netlists (circuits) into smaller subgraphs, aiding independent placement. As the optimal partition count is frequently unknown in advance, experimenting with different partition numbers is a common practice. Hence, it is a common practice to experiment with different partition counts and evaluate their impact on the partitioning objective. While the existing methods [10, 28, 37] assume that the number of partitions (k) is known beforehand, our proposed method can generalize to any k.
- **Non-generalizability to different cut functions:** Multiple objective functions for graph cut have been studied in the partitioning literature. The optimal objective function hinges upon the subsequent application in question. For instance, the two most relevant studies, DMoN [37] and GAP [28] focus on maximizing modularity and minimizing normalized cut respectively. Our framework is generic and can solve different partitioning objectives.
- **Assumption of differential objective function:** Existing neural approaches assume the objective function to be differentiable. As we illustrate in § 2, the assumption does not always hold in the real-world. For instance, the sparsest cut [6] and balanced cut [28] objectives are not differentiable.

1.2 Contributions:

In this paper, we circumvent the above-mentioned limitations through the following key contributions.

- **Versatile objectives:** We develop NeuroCUT; an auto-regressive, graph reinforcement learning framework that integrates positional information, to solve the graph partitioning problem for attributed graphs. Diverging from conventional algorithms, NeuroCUT can solve multiple partitioning objectives. Moreover, unlike other neural methods, NeuroCUT can accommodate diverse partitioning objectives, without the necessity for differentiability.
- **Inductivity to number of partitions:** The parameter space of NeuroCUT is independent of the partition count. This innovative decoupled architecture endows NeuroCUT with the ability to generalize effectively to arbitrary partition count specified during inference.
- **Empirical Assessment:** We perform comprehensive experiments employing real-world datasets, evaluating NeuroCUT across four distinct graph partitioning objectives. Our empirical investigation substantiates the efficacy of NeuroCUT in partitioning tasks, showcasing its robustness across a spectrum of objective functions. We also demonstrate the capacity of NeuroCUT to generalize effectively to partition sizes that it has not encountered during training.

2 PROBLEM FORMULATION

In this section, we introduce the concepts central to our work and formulate the problem. All the notations used in this work are outlined in Table 1.

Definition 1 (Graph). *We denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X})$ where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges and $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times |F|}$ refers to node feature matrix where F is the set of all features in graph $\mathcal{G}$.*

Definition 2 (Cut). *A cut $\mathcal{C} = (\mathcal{S}, \mathcal{T})$ is a partition of $\mathcal{V}$ into two subsets $\mathcal{S}$ and $\mathcal{T}$. The cut-set of $\mathcal{C} = (\mathcal{S}, \mathcal{T})$ is the set $\{(u, v) \in \mathcal{E} | u \in \mathcal{S}, v \in \mathcal{T}\}$ of edges that have one endpoint in $\mathcal{S}$ and the other endpoint in $\mathcal{T}$.*

Definition 3 (Graph Partitioning). *Given graph $\mathcal{G}$, we aim to partition $\mathcal{G}$ into k disjoint sets $\mathcal{P} = \{P_1, P_2, \cdots, P_k\}$ such that the union of the nodes in those sets is equal to $\mathcal{V}$ i.e $\cup_{i=1}^{k} P_i = \mathcal{V}$ and each node belongs to exactly one partition.*

Partitioning Objective: We aim to minimize/maximize a partitioning objective of the form $Obj(\mathcal{G}, \mathcal{P})$. A wide variety of objectives for graph partitioning have been proposed in the literature. Without loss of generality, we consider the following four objectives. These objectives are chosen due to being well studied in the literature, while also being diverse from each other. [1]

(1) *k-MinCut [32]:* Partition a graph into k partitions such that the total number of edges across partitions is minimized.

$$k\text{-mincut}(\mathcal{P}) = \sum_{l=1}^{|\mathcal{P}|} \frac{|\operatorname{cut}\left(P_l, \overline{P_l}\right)|}{\sum_{e \in \mathcal{E}} |e|} \tag{1}$$

Here P_l refers to the set of elements in l^{th} partition of $\mathcal{P}$ as described in Def. 3 and $\overline{P_l}$ refers to set of elements not in P_l.

(2) *Normalized Cut [34]:* The k-MinCut criteria favors cutting small sets of isolated nodes in the graph. To avoid this unnatural bias for partitioning out small sets of points, normalised cut computes the cut cost as a fraction of the total edge connections to all the nodes in the graph.

$$\text{Ncut}(\mathcal{P}) := \sum_{l=1}^{|\mathcal{P}|} \frac{|\operatorname{cut}\left(P_l, \overline{P_l}\right)|}{\operatorname{vol}\left(P_l, \mathcal{V}\right)} \tag{2}$$

Here, $\operatorname{vol}\left(P_l, \mathcal{V}\right) := \sum_{v_i \in P_l, v_j \in \mathcal{V}} e(v_i, v_j)$.

(3) *Balanced Cuts [28]:* Balanced cut favours partitions of equal sizes so an extra term that indicates the squared distance from equal sized partition is added to normalized cuts.

$$\text{Balanced-Cuts}(\mathcal{P}) := \sum_{l=1}^{|\mathcal{P}|} \frac{|\operatorname{cut}\left(P_l, \overline{P_l}\right)|}{\operatorname{vol}\left(P_l, \mathcal{V}\right)} + \frac{(|P_l| - |\mathcal{V}|/k)^2}{|\mathcal{V}|^2} \tag{3}$$

Here, $\operatorname{vol}\left(P_l, \mathcal{V}\right) := \sum_{v_i \in P_l, v_j \in \mathcal{V}} e(v_i, v_j)$

[1] Our framework is not restricted to these objectives.

Symbol	Meaning
$\mathcal{G}$	Graph
$\mathcal{V}$	Node set
e	Edge $e \in \mathcal{E}$
$\mathcal{E}$	Edge set
$\boldsymbol{X}$	Feature matrix containing raw node features
$\mathcal{N}_v$	Neighboring nodes of node v
$Obj(\mathcal{G}, \mathcal{P})$	Objective function based upon graph $\mathcal{G}$ and its partitioning $\mathcal{P}$
k	Number of partitions
$\mathcal{P}^t$	Partitioning at time t
P_i^t	i^{th} partition at time t
$\overline{P_i}$	Set of nodes that are not in the i^{th} partition
$\mathcal{S}^t$	State of system at step t
PART$(\mathcal{P}^t, v)$	Partition of node v at time t
$\mathcal{S}^t$	State representation of Partitions and Graph at step t
$\mathbf{pos}(v)$	Positional embedding of node v
$\mathbf{emb}_{\text{init}}(v)$	Initial embedding of node v
α	Number of anchor nodes for lipschitz embedding
T	Length of trajectory
π	Policy function

Table 1: Notations used in the paper

(4) Sparsest Cuts [6]: Two-way sparsest cuts minimize the cut edges relative to the number of nodes in the smaller partition. We generalize it to k-way sparsest cuts by summing up the value for all the partitions. The intuition behind sparsest cuts is that any partition should neither be very large nor very small.

$$\text{Sparsest-Cuts}(\mathcal{P}) = \sum_{l=1}^{|\mathcal{P}|} \phi(P_l, \overline{P_l}) \tag{4}$$

$$where \quad \phi(S, \bar{S}) = \frac{\text{cut}(S, \bar{S})}{\min(|S|, |\bar{S}|)} \tag{5}$$

Problem 1 (Learning to Partition Graph). *Given a graph $\mathcal{G}$ and the number of partitions k, the goal is to find a partitioning $\mathcal{P}$ of the graph $\mathcal{G}$ that optimizes a target objective function $Obj(\mathcal{G}, \mathcal{P})$. Towards this end, we aim to learn a policy π that assigns each node $v \in \mathcal{V}$ to a partition in $\mathcal{P}$.*

In addition to our primary goal of finding a partitioning that optimizes a certain objective function, we also desire policy π to have the following properties:

(1) **Inductive:** Policy π is inductive if the parameters of the policy are independent of both the size of the graph and the number of partitions k. If the policy is not inductive then it will be unable to infer on unseen size graphs/number of partitions.
(2) **Learning Versatile Objectives:** To optimize the parameters of the policy, a target objective function is required. The optimization objective may not be differentiable and it might not be always possible to obtain a differentiable formulation. Hence, the policy π should be capable of learning to optimize for a target objective that may or may not be differentiable.

3 NEUROCUT: PROPOSED METHODOLOGY

Fig. 1 describes the framework of NeuroCUT. For a given input graph $\mathcal{G}$, we first construct the initial partitions of nodes using a clustering based approach. Subsequently, a message-passing Gnn embeds the nodes of the graph ensuring inductivity to different graph sizes. Next, the assignment of nodes to partitions proceeds in a two-phased strategy. We first select a node to change its partition and then we choose a suitable partition for the selected node. Further, to ensure inductivity on the number of partitions, we *decouple* the number of partitions from the direct output representations of the model. This decoupling allows us to query the model to an unseen number of partitions. After a node's partition is updated, the *reward* with respect to change in partitioning objective value is computed and the parameters of the policy are optimized. The reward is learned through *reinforcement learning (RL)* [35].

The choice of using RL is motivated through two observations. Firstly, cut problems on graphs are generally recognized as NP-hard, making it impractical to rely on ground-truth data, which would be computationally infeasible to obtain. Secondly, the cut objective may lack differentiability. Therefore, it becomes essential to adopt a learning paradigm that can be trained even under these non-differentiable constraints. In this context, RL effectively addresses both of these critical requirements. Additionally, in the process of sequentially constructing a solution, RL allows us to model the gain obtained by perturbing the partition of a node.

Markov Decision Process. Given a graph $\mathcal{G}$, our objective is to find the partitioning $\mathcal{P}$ that maximizes/minimizes the target objective function $Obj(\mathcal{G}, \mathcal{P})$. We model the task of iteratively updating the partition for a node as a *Markov Decision Process (MDP)* defined by the tuple $(S, \mathcal{A}, \rho, R, \gamma)$. Here, S is the *state space*, $\mathcal{A}$ is the set of all possible *actions*, $\rho : S \times S \times \mathcal{A} \to [0, 1]$ denotes the *state transition probability function*, $R : S \times \mathcal{A} \to \mathbb{R}$ denotes the *reward function* and $\gamma \in (0, 1)$ the *discounting factor*. We next formalize each of these notions in our MDP formulation.

3.1 State: Initialization & Positional Encoding

Initialization. Instead of directly starting from empty partitions, we perform a warm start operation that clusters the nodes of the graph to obtain the initial graph partitions. The graph is clustered into k clusters based on their raw features and positional embeddings (discussed below). We use *K-means* [25] algorithm for this task. The details of the clustering are present in Appendix A.2.

Positional Encoding (Embeddings). Given that the partitioning objectives are NP-hard mainly due to the combinatorial nature of the graph structure, we look for representations that capture the location of a node in the graph. Positional encodings provide an idea of the position in space of a given node within the graph. Two nodes that are closer in the graph, should be closer in the embedding space. Towards this, we use *Lipschitz Embedding* [5]. Let $\mathcal{A} = \{a_1, \cdots, a_\alpha\} \subseteq \mathcal{V}$ be a randomly selected subset of α nodes. We call them anchor nodes. From each anchor node i, the walker starts a random walk [4] and jumps to a neighboring node j with a transition probability $(\mathbf{W}_{ij})$ governed by the transition probability matrix $\mathbf{W} \in \mathbb{R}^{|V| \times |V|}$. Furthermore, at each step, with probability c the walker jumps to a neighboring node j and returns to the node i with $1 - c$. Let $\vec{r}_{ij}$ corresponds to the probability of the random walker starting from node i and reaching node j.

$$\vec{r}_i = c\tilde{\mathbf{W}}\vec{r}_i + (1 - c)\vec{e}_i \tag{6}$$

Eq. 6 describes the random walk starting at node i. In vector $\vec{e}_i \in \mathbb{R}^{|V| \times 1}$, only the i^{th} element (the initial anchor node) is 1, and the

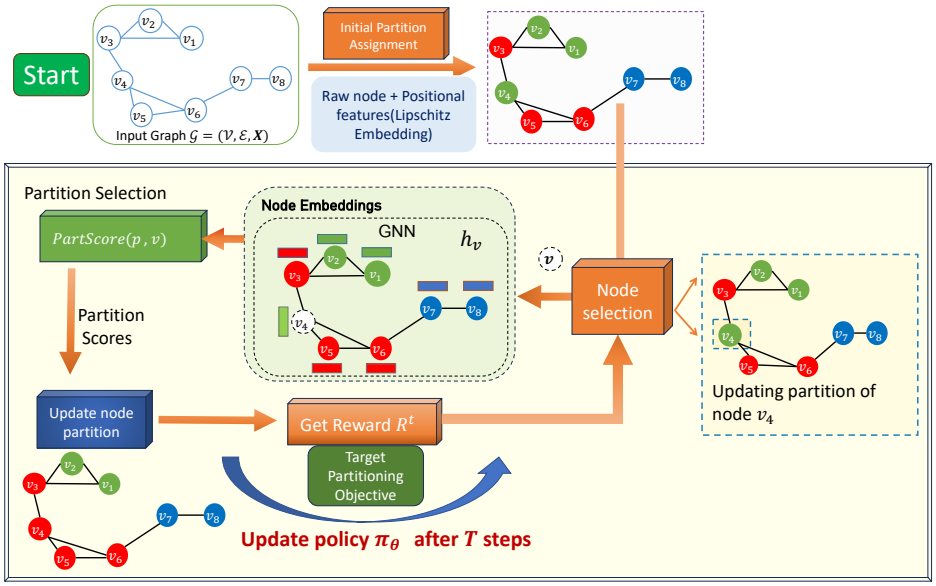


Figure 1: Architecture of NeuroCUT. First, the initial partitioning of the graph is performed based on node raw features and positional embeddings. These embeddings are refined using GNN to infuse toplogical information from neighborhood. At each step a node is selected and its partitioned is updated. During the training phase the GNN parameters are updated and hence embeddings are re-computed. During inference, the GNN is called only once to compute the embeddings of the nodes of the graph.

rest are set to zero. We set $\mathbf{W}_{ij} = \frac{1}{degree(j)}$ if edge $e_{ij} \in \mathcal{E}$, 0 otherwise. The random walk with restart process is repeated for β iterations, where β is a hyper-parameter. Here $\vec{e}_i \in \mathbb{R}^{|V|\times 1}$ and c is a scalar.

Based upon the obtained random walk vectors for the set of anchor nodes $\mathcal{A}$, we embed all nodes $u \in \mathcal{V}$ in a α-dimensional feature space:

$$\mathbf{pos}(u) = [r_{1u}, r_{2u}, \cdots, r_{\alpha u}] \tag{7}$$

To accommodate both raw feature and positional information we concatenate the raw node features i.e $\mathbf{X}[u]$ with the positional embedding $\mathbf{pos}(u)$ for each node u to obtain the *initial embedding* which will act as input to our neural model. Specifically,

$$\mathbf{emb}_{\mathbf{init}}(u) = \mathbf{X}[u] \quad \| \quad \mathbf{pos}(u) \tag{8}$$

In the above equation, $\|$ represents the concatenation operator.

State. The state space characterizes the state of the system at time t in terms of the current set of partitions $\mathcal{P}^t$. Intuitively the state should contain information to help our model make a decision to select the next node and the partition for the node to be assigned. Let $\mathcal{P}^t$ denote the status of partitions at time t wherein a partition P_i^t is represented by all nodes belonging to i^{th} partition. The state of the system at step t is defined as

$$S^t = \{S_1^t, S_2^t, \cdots S_k^t : \ S_i^t = \{\mathbf{emb}_{\mathbf{init}}(v) \ \ \forall v \in P_i^t\}\} \tag{9}$$

Here state of each partition is represented by the collection of *initial embedding* of nodes in that partition.

3.2 Action: *Selection of Nodes* and *Partitions*

Towards finding the best partitioning scheme for the target partition objective we propose a ***2-step action*** strategy to update the partitions of nodes. The first phase consists of identifying a node to update its partition. Instead of arbitrarily picking a node, we propose to prioritize selecting nodes for which the new assignment is more likely to improve the overall partitioning objective. In comparison to a strategy that arbitrarily selects nodes, the above mechanism promises greater improvement in the objective with less number of iterations. In the second phase, we calculate the score of each partition $\mathcal{P}$ with respect to the selected node from the first phase and then assign it to one of the partitions based upon the partitioning scores. We discuss both these phases in details below.

3.2.1 *Phase 1: Node selection to identify node to perturb.*

Let $\text{PART}(\mathcal{P}^t, v)$ denote the partition of the node v at step t. Our proposed formulation involves selecting a node v at step t belonging to partition $\text{PART}(\mathcal{P}^t, v)$ and then assigning it to a new partition. The newly assigned partition and the current partition of the node could also be same.

Towards this, we design a heuristic to prioritize selecting nodes which when placed in a new partition are more likely to improve the overall objective value. A node v is highly likely to be moved from its current partition if most of its neighbours are in a different partition than that of node v. Towards this, we calculate the score of nodes $v \in \mathcal{V}$ in the graph as the ratio between the maximum number of neighbors in another partition and the number of neighbors in the same partition as v. Intuitively, if a partition exists in which the majority of neighboring nodes of a given node v belong, and yet node v is not included in that partition, then there is a high probability that node v should be subjected to perturbation. Specifically the score of node v at step t is defined as:

$$\text{NodeScore}^t[v] = \frac{\max_{p \in P^t - \text{PART}(P^t, v)} |u | u \in \mathcal{N}(v) \ni \text{PART}(P^t, u) = p|}{|u | u \in \mathcal{N}(v) \ni \text{PART}(P^t, u) = \text{PART}(P^t, v)|} \times \frac{1}{degree(v)} \tag{10}$$

For a node of interest v, the numerator computes the maximum number of neighbors in a different partition than v. As described in tab. 1, PART(P^t, v) refers to partition of v at step t. The expression P^t − PART(P^t, v) computes all other partitions than partition of v. The term $|u|u \in \mathcal{N}(v) \ni \text{PART}(P^t, u) = p|$ computes the number of neighbors of v which are in a partition p. The denominator computes the number of neighbors of node v in the same partition as v, refered to as PART(P^t, v). Further, a node having a higher degree implies it has several edges associated to it, hence, an incorrect placement of it could contribute to higher partitioning value. Hence, we normalize the scores by the *degree* of the node.

3.2.2 *Phase 2: Inductive Method for Partition Selection.*

Once a node is selected, the next phase involves choosing the new partition for the node. Towards this, we design an approach empowered by Graph Neural Networks (GNNs) [12] which enables the model to be inductive with respect to size of graph. Further, instead of predicting a fixed-size score vector [28, 38] for the number of partitions, our proposed method of computing partition scores allows the model to be inductive to the number of partitions too. We discuss both above points in section below.

Message passing through Graph Neural Network

To capture the interaction between different nodes and their features along with the graph topology, we parameterize our policy by a Graph Neural Network (GNN) [12]. GNNs combine node feature information and the graph structure to learn better representations via feature propagation and aggregation.

We first initialize the input layer of each node $u \in \mathcal{V}$ in graph as $\mathbf{h}_u^0 = \mathbf{emb}_{\text{init}}(u)$ using eq. 8. We perform L layers of message passing to compute representations of nodes. To generate the embedding for node u at layer $l+1$ we perform the following transformation[12]:

$$\mathbf{h}_u^{l+1} = \mathbf{W}_1^l \mathbf{h}_u^l + \mathbf{W}_2^l \cdot \frac{1}{|\mathcal{N}_u|} \sum_{u' \in \mathcal{N}_u} \mathbf{h}_{u'}^l \tag{11}$$

where $\mathbf{h}_u^{(l)}$ is the node embedding in layer l. $\mathbf{W}_1^l$ and $\mathbf{W}_2^l$ are trainable weight matrices at layer l.

Following L layers of message passing, the final node representation of node u in the L^{th} layer is denoted by $\mathbf{h}_u^L \in \mathbb{R}^d$. Intuitively $\mathbf{h}_u^L$ characterizes u using a combination of its own features and features aggregated from its neighborhood.

Scoring partitions. Recall from eq. 9, each partition at time t is represented using the nodes belonging to that partition. Building upon this, we compute the score of each partition $p \in \mathcal{P}^t$ with respect to the node v selected in *Phase 1* using all the nodes in p. In contrast to predicting a fixed-size score vector corresponding to number of partitions [28, 38], the proposed design choice makes the model inductive to the number of partitions. Specifically, the number of partitions are not directly tied to the output dimensions of the neural model.

Having obtained the transformed node embeddings through a Gnn in Eq. 11, we now compute the (unnormalized) score for node v selected in phase 1 for each partition $p \in \mathcal{P}^t$ as follows:

$$\begin{aligned}\text{PartScore}(p, v) = \text{AGG}(\{&\text{MLP}(\sigma(h_v|h_u)) \\ &\forall u \in \mathcal{N}(v) \ni \text{PART}(\mathcal{P}^t, u) = p\})\end{aligned} \tag{12}$$

The above equation concatenates the selected node v's embedding with its neighbors $u \in \mathcal{N}(v)$ that belong to the partition p under consideration. In general, the strength of a partition assignment to a node is higher if its neighbors also belong to the same partition. The above formulation surfaces this strength in the embedding space. The concatenated representation $(h_v|h_u)\forall u \in \mathcal{N}(v)$ is passed through an MLP that converts the vector into a score (scalar). We then apply an aggregation operator (e.g., mean) over all neighbors of u belonging to $\mathcal{P}^t$ to get an unnormalized score for partition p. Here σ is an activation function.

To compute the normalized score at step t is finally calculated as softmax over the all partitions $p \in \mathcal{P}^t$ for the currently selected node v. Mathematically, the probability of taking action $a^t{=}p$ at time step t at state S^t is defined as:

$$\pi((a^t{=}p)/S^t) = \frac{\exp\left(\text{PartScore}\left(p, v\right)\right)}{\sum_{p' \in \mathcal{P}^t} \exp\left(\text{PartScore}\left(p', v\right)\right)} \tag{13}$$

During the course of trajectory of length T, we sample action $a^t \in \mathcal{P}$ i.e., the assignment of the partition for the node selected in phase 1 at step t using policy π.

State Transition. After action a^t is applied at state S^t, the state is updated to S^{t+1} that involves updating the partition set $\mathcal{P}^{t+1}$. Specifically, if node v belonged to i^{th} partition at time t and its partition has been changed to j in phase 2, then we apply the below operations in order.

$$P_i^{t+1} \leftarrow P_i^t \setminus v \text{ and } P_j^{t+1} \leftarrow P_j^t \cup v \tag{14}$$

3.3 Reward, Training & Inference

Reward. Our aim is to improve the value of the overall partitioning objective. One way is to define the reward R^t at step $t \geq 0$ as the change in objective value of the partitioning i.e $Obj(\mathcal{G}, \mathcal{P}^t)$ at step t, i.e., $R^t = (Obj(\mathcal{G}, \mathcal{P}^{t+1}) - Obj(\mathcal{G}, \mathcal{P}^t))/(Obj(\mathcal{G}, \mathcal{P}^{t+1}) + Obj(\mathcal{G}, \mathcal{P}^t))$. Here, the denominator term ensures that the model receives a significant reward for even slight enhancements when dealing with a small objective value. However, this definition of reward focuses on short-term improvements instead of long-term. Hence, to prevent this local greedy behavior and to capture the combinatorial aspect of the selections, we use *discounted rewards* D^t to increase the probability of actions that lead to higher rewards in the long term [35]. The discounted rewards are computed as the sum of the rewards over a *horizon* of actions with varying degrees of importance (short-term and long-term). Mathematically,

$$D^t = R^t + \gamma R^{t+1} + \gamma^2 R^{t+2} + \ldots = \sum_{j=0}^{T-t} \gamma^j R^{t+j} \tag{15}$$

where T is the length of the horizon and $\gamma \in (0, 1]$ is a *discounting factor* (hyper-parameter) describing how much we favor immediate rewards over the long-term future rewards.

The above reward mechanism provides flexibility to our framework to be versatile to objectives of different nature, that may or may not be differentiable. This is an advantage over existing neural methods [28, 38] where having a differentiable form of the partitioning objective is a pre-requisite.

Policy Loss Computation and Parameter Update. Our objective is to learn parameters of our policy network in such a way that actions that lead to an overall improvement of the partitioning objective are favored more over others. Towards this, we use *REINFORCE gradient estimator* [40] to optimize the parameters of our policy network. Specifically, we wish to maximize the reward

obtained for the horizon of length T with discounted rewards D^t. Towards this end, we define a reward function $J(\pi_\theta)$ as:

$$J(\pi_\theta) = \mathbb{E}\Big[\sum_{t=0}^{T} (D^t)\Big] \tag{16}$$

We, then, optimize $J(\pi_\theta)$ via policy gradient [35] as follows:

$$\nabla J(\pi_\theta) = \left[\sum_{t=0}^{T} (D^t)\, \nabla_\theta log \pi_\theta(a^t/S^t)\right] \tag{17}$$

Training and Inference. For a given graph, we optimize the parameters of the policy network π_θ for T steps. Note that the trajectory length T is not kept very large to avoid the long-horizon problem. During inference, we compute the initial node embeddings, obtain initial partitioning and then run the forward pass of our policy to improve the partitioning objective over time.

3.4 Time complexity

The time complexity of NeuroCUT during inference is $O\big((\alpha \times \beta \times |\mathcal{E}|) + (|\mathcal{E}| + k) \times T'\big)$. Here α is the number of anchor nodes, β is number of random walk iterations, k is the number of partitions and T' is the number of iterations during inference. Typically α, β and k are $<<$ $|\mathcal{V}|$ and $T' = o(|\mathcal{V}|)$. Further, for sparse graphs $|\mathcal{E}| = O(|\mathcal{V}|)$. Hence time complexity of NeuroCUT is $o(|\mathcal{V}|^2)$. For detailed derivation please see appendix A.4.

4 EXPERIMENTS

In this section, we demonstrate the efficacy of NeuroCUT against state-of-the-art methods and establish that:

- **Efficacy and robustness:** NeuroCUT produces the best results over diverse partitioning objective functions. This establishes the robustness of NeuroCUT.
- **Inductivity:** As one of the major strengths, unlike existing neural models such as GAP [28], DMon [38] and MinCutPool [3], our proposed method NeuroCUT is inductive on the number of partitions and consequently can generalize to unseen number of partitions.

4.1 Experimental Setup

4.1.1 Datasets: We use *four real datasets* for our experiments. They are described below and their statistics are present in the appendix (please see Table 7).

- **Cora** and **Citeseer** [33]: These are citation networks where nodes correspond to individual papers and edges represent citations between papers. The node features are extracted using a bag-of-words approach applied to paper abstracts.
- **Harbin** [24]: This is a road network extracted from Harbin city, China. The nodes correspond to road intersections and node features represent latitude and longitude of a road intersection.
- **Actor** [30]: This dataset is based upon Wikipedia data where each node in the graph corresponds to an actor, and the edge between two nodes denotes co-occurrence on the same Wikipedia page. Node features correspond to keywords in Wikipedia pages.

4.1.2 Partitioning objectives: We evaluate our method on a diverse set of *four partitioning objectives* described in Section 2, namely *Normalized Cut, Balanced Cut, k-MinCut, and Sparsest Cut.* In addition to evaluating on diverse objectives, we also choose a diverse number of partitions for evaluation, specifically, $k = 2,\ 5$ and 10.

4.1.3 Baselines: We compare our proposed method with both *neural* as well as *non-neural* methods.

Neural baselines: We compare with DMon [38], GAP [28], MinCutPool [3] and Ortho [38]. DMon is the state-of-the-art neural attributed-graph clustering method. GAP is optimized for balanced normalized cuts with an end-to-end framework with a differentiable loss function which is a continuous relaxation version of normalized cut. MinCutPool optimizes for normalized cut and uses an additional orthogonality regularizer. Ortho is the orthogonality regularizer described in DMon and MinCutPool.

Non-neural baselines: Following the settings of DMon [38], we compare with K-means clustering applied on raw node features. We also compare with standard graph clustering methods hMetis [17] and Spectral clustering [29] in App Sec. A.5.

4.1.4 Other settings: We run all our experiments on an Ubuntu 20.04 system running on Intel Xeon 6248 processor with 96 cores and 1 NVIDIA A100 GPU with 40GB memory for our experiments. For NeuroCUT we used GraphSage[12] as our GNN with number of layers $L = 2$, learning rate as 0.0001, hidden size = 32. We used Adam optimizer for training the parameters of our policy network π_θ. For computing discounted reward in RL, we use discount factor $\gamma = 0.99$. We set the length of trajectory T during training as 2. At time step t, the rewards are computed from time t to $t + T$ and parameters of the policy π_θ are updated. The default number of anchor nodes for computing positional embeddings is set to 35. We set $\beta = 100$ and $c = 0.85$ in Eq. 6.

Dataset	Method	$k = 2$	$k = 5$	$k = 10$
Cora	Kmeans	0.65	3.26	7.44
	MinCutPool	0.12	0.61	1.65
	DMon	0.57	3.07	7.40
	Ortho	0.80	1.88	4.06
	GAP	0.10	0.68	-
	NeuroCUT	**0.02**	**0.33**	**0.92**
CiteSeer	Kmeans	0.30	2.35	5.21
	MinCutPool	0.10	0.38	1.04
	DMon	0.33	2.71	6.84
	Ortho	0.28	1.51	3.25
	GAP	0.12	-	-
	NeuroCUT	**0.02**	**0.20**	**0.44**
Harbin	Kmeans	0.56	2.34	5.40
	MinCutPool	-	-	-
	DMon	0.98	3.25	-
	Ortho	-	-	-
	GAP	0.25	-	-
	NeuroCUT	**0.01**	**0.07**	**0.28**
Actor	Kmeans	0.99	4.00	8.98
	MinCutPool	0.55	1.97	4.73
	DMon	0.77	3.46	8.08
	Ortho	1.05	3.98	8.90
	GAP	0.20	-	-
	NeuroCUT	**0.17**	**0.99**	**4.66**

Table 2: Results on Normalized Cut. Our model NeuroCUT produces the best (lower is better) performance across all datasets and the number of partitions k.

4.2 Results on Transductive Setting

In the transductive setting, we compare our proposed method NeuroCUT against the mentioned baselines, where the neural models are trained and tested with the same number of partitions. Tables 2-5 present the results for different partitioning objectives

Dataset	Method	$k=2$	$k=5$	$k=10$
Cora	Kmeans	3.41	14.90	32.52
	MinCutPool	0.52	2.44	6.68
	DMon	0.45	1.89	5.82
	Ortho	2.73	7.06	15.70
	GAP	0.41	2.80	-
	NeuroCUT	**0.13**	**1.46**	**3.03**
CiteSeer	Kmeans	1.53	13.70	22.20
	MinCutPool	0.31	1.32	3.62
	DMon	0.35	1.21	3.62
	Ortho	0.88	4.15	10.60
	GAP	0.60	-	-
	NeuroCUT	**0.11**	**0.49**	**1.19**
Harbin	Kmeans	1.91	7.44	17.03
	MinCutPool	-	-	-
	DMon	2.01	-	-
	Ortho	-	-	-
	GAP	1.56	-	-
	NeuroCUT	**0.06**	**0.23**	**0.82**
Actor	Kmeans	5.43	19.40	40.34
	MinCutPool	2.44	8.51	19.81
	DMon	1.71	9.50	21.01
	Ortho	3.42	16.55	40.4
	GAP	1.35	-	-
	NeuroCUT	**0.65**	**2.04**	**2.88**

Table 3: Results on Sparsest Cut. Our model NeuroCUT produces the best (lower is better) performance across all datasets and number of partitions k.

Dataset	Method	$k=2$	$k=5$	$k=10$
Cora	Kmeans	0.32	0.34	0.61
	MinCutPool	0.06	0.12	0.17
	DMon	0.05	0.09	0.14
	Ortho	0.33	0.34	0.38
	GAP	0.05	0.10	0.11
	NeuroCUT	**~ 0.0**	**~ 0.0**	**0.06**
CiteSeer	Kmeans	0.12	0.27	0.43
	MinCutPool	0.04	0.08	0.10
	DMon	0.05	0.07	0.12
	Ortho	0.13	0.47	0.30
	GAP	0.05	0.08	0.10
	NeuroCUT	**0.01**	**0.02**	**0.03**
Harbin	Kmeans	0.28	0.46	0.54
	MinCutPool	**0.0**	**0.0**	**0.0**
	DMon	0.10	0.01	~ 0.0
	Ortho	**0.0**	**0.0**	**0.0**
	GAP	0.01	0.01	0.01
	NeuroCUT	~ 0.0	0.01	0.03
Actor	Kmeans	0.48	0.54	0.80
	MinCutPool	0.25	0.35	0.42
	DMon	0.17	0.39	0.44
	Ortho	0.35	0.65	0.83
	GAP	0.09	0.12	0.26
	NeuroCUT	**~ 0.0**	**~ 0.0**	**~ 0.0**

Table 4: Results on k-MinCut. In most of the cases, our model NeuroCUT produces the best (lower is better) performance across all datasets and the number of partitions k. Values less than 10^{-2} are approximated to 0.

Dataset	Method	$k=2$	$k=5$	$k=10$
Cora	Kmeans	0.68	3.90	7.44
	MinCutPool	0.13	**0.60**	1.62
	DMon	0.11	0.48	1.47
	Ortho	0.80	1.89	4.06
	GAP	**0.10**	0.74	-
	NeuroCUT	0.45	0.64	**1.08**
CiteSeer	Kmeans	0.42	2.64	5.30
	MinCutPool	0.09	0.38	1.04
	DMon	0.10	0.37	1.1
	Ortho	0.28	1.51	3.32
	GAP	0.23	-	-
	NeuroCUT	**0.07**	**0.24**	**0.60**
Harbin	Kmeans	0.56	2.37	5.41
	MinCutPool	-	-	-
	DMon	1.30	-	-
	Ortho	-	-	-
	GAP	0.72	-	-
	NeuroCUT	**0.21**	**0.11**	**0.27**
Actor	Kmeans	1.00	4.06	9.08
	MinCutPool	0.55	1.96	4.71
	DMon	0.34	2.01	4.80
	Ortho	1.05	3.90	8.91
	GAP	**0.25**	-	-
	NeuroCUT	0.59	**1.69**	**4.42**

Table 5: Results on Balanced Cut. In most cases, our model NeuroCUT produces the best (lower is better) performance across all datasets and number of partitions k.

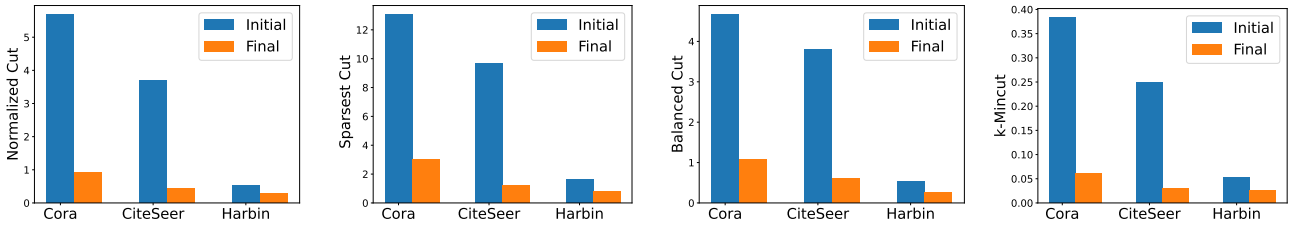
across all datasets and methods. For the objectives under consideration, a smaller value depicts better performance. NeuroCUT demonstrates superior performance compared to both neural and non-neural baselines across four distinct partitioning objectives, highlighting its robustness. Further, we would also like to point out that in many cases, existing baselines produce *"nan"* values which are represented by "-" in the result tables. This is due to the reason that every partition is not assigned atleast one node which leads to the situation where denominator becomes 0 in *normalized*, *balanced* and *sparsest* cut objectives as defined in Sec. 2.

Our method NeuroCUT incorporates dependency during inference and this leads to robust performance. Specifically, as the architecture is auto-regressive, it takes into account the current state before moving ahead as opposed to a single shot pass in methods such as GAP [28]. Further, unlike other methods, NeuroCUT also incorporates positional information in the form of *Lipschitz embedding* to better contextualize global node positional information. We also observe that the non-neural method K-means fails to perform on all objective functions since its objective is not aligned with the main objectives under consideration.

4.3 Results on Inductivity to Partition Count

As detailed in Section 3.2.2, the decoupling of parameter size and the number of partitions allows NeuroCUT to generalize effectively to an unseen number of partitions. In this section, we empirically analyze the generalization performance of NeuroCUT to an unknown number of partitions. We compare it with the non-neural baseline K-means, as the neural methods like GAP, DMon, Ortho, and MinCutPool cannot be employed to infer on an unseen number of partitions. In Table 6, we present the results on inductivity. Specifically, we trained NeuroCUT on different partition sizes ($k=5$ and $k=8$) and tested it on an unseen partition size $k=10$. The inductive version, referred to as NeuroCUT-I[2], obtains high-quality results on different datasets. Note that while the non-neural method such as K-means needs to be re-run for the

[2]For clarity purposes, in this experiment we use NeuroCUT-T to refer to the transductive setting whose results are presented in Sec. 4.2



(a) Normalized Cut (b) Sparsest Cut (c) Balanced Cut (d) k-MinCut

Figure 2: Results on the initial warm start and the final cut values obtained by NeuroCUT at k=10. It shows that our neural model NeuroCUT (Final) performs more accurate node and partition selection to optimize the objective function. Subsequently, there is a significant difference between the initial and final cut values.

Dataset	Method	Normalized	Sparsest	k-MinCut	Balanced
Cora	Kmeans	7.44	32.52	0.61	7.53
	NeuroCUT-T	0.92	3.03	0.06	1.08
	NeuroCUT-I	1.18	5.04	0.02	2.03
CiteSeer	Kmeans	5.21	22.20	0.43	5.30
	NeuroCUT-T	0.44	1.19	0.03	0.60
	NeuroCUT-I	0.63	1.62	0.03	0.62
Harbin	Kmeans	5.40	17.03	0.54	5.41
	NeuroCUT-T	0.28	0.82	0.03	0.27
	NeuroCUT-I	0.27	0.87	0.03	0.29

Table 6: Inductivity to unseen partition count. Here we set the target number of partitions $k = 10$. I stands for inductive and T for transductive setting. In this table, the transductive version of NeuroCUT trained on $k = 10$ serves as a point of reference when assessing the quality of the inductive version of NeuroCUT.

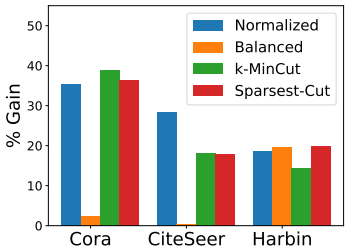


Figure 3: Node Selection in *Phase 1* with Heuristic vs Random: Relative % improvement (gain) in cut values obtained by NeuroCUT when using *node selection heuristic* against *random node selection* for $k = 5$. In most cases, our heuristic finds significantly better cuts than a random node selection procedure.

unseen partitions, NeuroCUT only needs to perform forward pass to produce the results on an unseen partition size.

4.4 Ablation Studies

Initial Warm Start vs Final Cut Values. NeuroCUT takes a warm start by partitioning nodes based on clustering over their raw node features and Lipschitz embeddings, which are subsequently fine-tuned auto-regressively through the *phase 1* and *phase 2* of NeuroCUT. In this section, we measure, how much the partitioning objective has improved since the initial clustering? Figure 2 sheds light on this question. Specifically, it shows the difference between the initial cut value after clustering and the final cut value from the partitions produced by our method. We note that there is a significant difference between the initial and final cut values, which essentially shows the effectiveness of NeuroCUT.

Impact of Node Selection Procedures. In this section, we explore the effectiveness of our proposed node selection heuristic in enhancing overall quality. Specifically, we measure the relative performance gain(in %) obtained by NeuroCUT when using the node selection heuristic as proposed in Section 3.2.1 over a random node selection strategy. Figure 3 shows the percentage gain for three datasets. We observe that a simpler node selection where we select all the nodes one by one in a random order, yields substantially inferior results in comparison to the heuristic proposed by us. This suggests that the proposed sophisticated node selection strategy plays a crucial role in optimizing the overall performance of NeuroCUT.

Impact of Clustering initialization in Warm-start Phase: In App. A.6 we study the impact of different initializations in the initial warm start phase.

5 CONCLUSIONS

In this work, we study the problem of graph partitioning with node features. Existing neural methods for addressing this problem require the target objective to be differentiable and necessitate prior knowledge of the number of partitions. In this paper, we introduced NeuroCUT, a framework to effectively address the graph partitioning problem with node features. NeuroCUT tackles these challenges using a reinforcement learning-based approach that can adapt to any target objective function. Further, attributed to its decoupled parameter space and partition count, NeuroCUT can generalize to an unseen number of partitions. The efficacy of our approach is empirically validated through an extensive evaluation on four datasets, four graph partitioning objectives and diverse partition counts. Notably, our method shows significant performance gains when compared to the state-of-the-art techniques, proving its competence in both inductive and transductive settings.

Limitations and Future Works: Achieving a sub-quadratic computation complexity for an inductive neural algorithm for attributed graph partitioning is an open challenge. We intend to pursue this research in our future endeavors.

REFERENCES

[1] Emmanuel Abbe. 2017. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research* 18, 1 (2017), 6446–6531.

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 475–486.

[3] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. 2020. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*. PMLR, 874–883.

[4] Monica Bianchini, Marco Gori, and Franco Scarselli. 2005. Inside pagerank. *ACM Transactions on Internet Technology (TOIT)* 5, 1 (2005), 92–128.

[5] Jean Bourgain. 1985. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics* 52 (1985), 46–52.

[6] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D Sivakumar. 2006. On the hardness of approximating multicut and sparsest-cut. *computational complexity* 15 (2006), 94–114.

[7] Fan Chung. 2007. Four proofs for the Cheeger inequality and graph partition algorithms. In *Proceedings of ICCM*, Vol. 2. Citeseer, 378.

[8] Chi Thang Duong, Thanh Tam Nguyen, Trung-Dung Hoang, Hongzhi Yin, Matthias Weidlich, and Quoc Viet Hung Nguyen. 2023. Deep MinCut: Learning Node Embeddings by Detecting Communities. *Pattern Recognition* 134 (2023), 109126.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231.

[10] Alice Gatti, Zhixiong Hu, Tess Smidt, Esmond G Ng, and Pieter Ghysels. 2022. Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks. *The Journal of Machine Learning Research* 23, 1 (2022), 13675–13702.

[11] Ralph E Gomory and Tien Chung Hu. 1961. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.* 9, 4 (1961), 551–570.

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[13] David Ireland and Giovanni Montana. 2022. Lense: Learning to navigate subgraph embeddings for large-scale combinatorial optimisation. In *International Conference on Machine Learning*. PMLR, 9622–9638.

[14] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227* (2019).

[15] Steffen Jung and Margret Keuper. 2022. Learning to solve minimum cost multicuts efficiently using edge-weighted graph convolutional neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 485–501.

[16] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. 2011. *VLSI physical design: from graph partitioning to timing closure*. Vol. 312. Springer.

[17] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1997. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings of the 34th annual Design Automation Conference*. 526–529.

[18] George Karypis and Vipin Kumar. 1998. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing* 48, 1 (1998), 96–129.

[19] George Karypis and Vipin Kumar. 1999. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE design automation conference*. 343–348.

[20] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems* 30 (2017).

[21] Wouter Kool, Herke Van Hoof, and Max Welling. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475* (2018).

[22] Jure Leskovec and Julian Mcauley. 2012. Learning to discover social circles in ego networks. *Advances in neural information processing systems* 25 (2012).

[23] Hao Li, Gary W Rosenwald, Juhwan Jung, and Chen-Ching Liu. 2005. Strategic power infrastructure defense. *Proc. IEEE* 93, 5 (2005), 918–933.

[24] Xiucheng Li, Gao Cong, Aixin Sun, and Yun Cheng. 2019. Learning travel time distributions with deep generative model. In *The World Wide Web Conference*. 1017–1027.

[25] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[26] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. 2020. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems* 33 (2020), 20000–20011.

[27] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research* 134 (2021), 105400.

[28] Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. 2019. Gap: Generalizable approximate graph partitioning framework. *arXiv preprint arXiv:1903.00614* (2019).

[29] Andrew Ng, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* 14 (2001).

[30] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. 2023. A critical look at the evaluation of GNNs under heterophily: are we really making progress? *arXiv preprint arXiv:2302.11640* (2023).

[31] Josep M Pujol, Vijay Erramilli, and Pablo Rodriguez. 2009. Divide and conquer: Partitioning online social networks. *arXiv preprint arXiv:0905.4918* (2009).

[32] Huzur Saran and Vijay V Vazirani. 1995. Finding k cuts within twice the optimal. *SIAM J. Comput.* 24, 1 (1995), 101–108.

[33] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[34] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

[35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[36] Amirmahdi Tafreshian and Neda Masoud. 2020. Trip-based graph partitioning in dynamic ridesharing. *Transportation Research Part C: Emerging Technologies* 114 (2020), 532–553.

[37] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2023. Graph clustering with graph neural networks. *Journal of Machine Learning Research* 24, 127 (2023), 1–21.

[38] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2023. Graph clustering with graph neural networks. *Journal of Machine Learning Research* 24, 127 (2023), 1–21.

[39] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 889–898.

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

[41] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. 2021. NeuroLKH: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems* 34 (2021), 7472–7483.

A APPENDIX

A.1 Additional Related Work

In recent years there has been a growing interest in using machine learning techniques to design heuristics for NP-Hard combinatorial problems on graphs. This line of work is motivated from the observation that traditional algorithmic approaches are limited to using the same heuristic irrespective of the underlying data distribution. Neural approaches, instead, focus on *learning* the heuristic as a function of the data without deriving explicit algorithms [8, 15, 28]. Significant advances have been made in learning to solve a variety of combinatorial optimization problems such as Set Cover, Travelling Salesman Problem, Influence Maximization etc. [14, 20, 21, 26]. The neural architectural designs in these methods are generally suitable for a specific set of CO problems [27] of similar nature. For instance, AM [21] and NeuroLKH [41] are targeted towards routing problems. GCOMB [26] and LeNSE [13] aim to solve problems that are submodular in nature. Owing to these assumptions, the architectural designs in these works are not suitable for our problem of k-way inductive graph partitioning with versatile partitioning objectives.

A.2 Clustering for Initialization

As discussed in sec 3.1 in main paper, we first cluster the nodes of the graph into k clusters where k is the number of partitions. Towards this, we apply *K-means* algorithm [25] on the nodes of the graph where a node is represented by its raw node features and positional representation i.e $\mathbf{emb}_{\mathbf{init}}(u)\ \forall u \in \mathcal{V}$ based upon eq. 8. Further, we used L_{inf} norm as the distance metric for clustering.

A.3 Dataset Statistics

Table 7 describes the statistics of datasets used in our work. For all datasets we use their largest connected component.

Table 7: Dataset Statistics

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	#Features	Average Degree	Clustering Coefficient	Degree Assortativity
Cora	2485	5069	1433	4.07	0.23	-0.07
CiteSeer	2120	3679	3703	3.47	0.169	0.0075
Harbin	6598	10492	2	3.18	0.036	0.22
Actor	6198	14879	931	4.82	0.05	-0.048
Facebook	1034	26749	-	51.7	0.526	0.431
SBM	500	5150	-	20.6	0.18	-0.0059

A.4 Time complexity analysis of NeuroCUT

We derive the time complexity of forward pass of NeuroCUT as discussed in sec. 3.4 in main paper.

(1) First the positional embeddings for all nodes in the graph are computed. This involves running RWR for α anchor nodes for β iterations (Eq. 6). This takes $O(\alpha \times \beta \times |\mathcal{E}|)$ time. **(2)** Next, GNN is called to compute embeddings of node. In each layer of GNN a node $v \in \mathcal{V}$ aggregates message from d neighbors where d is the average degree of a node. This takes $O(\mathcal{V})$ time. This operation is repeat for L layers. Since L is typically 1 or 2 hence we ignore this factor. **(3)** The node selection algorithm is used that computes score for each node based upon its neighborhood using eq. 10. This consumes $O(|\mathcal{V}| \times d)$ time. **(4)** Finally for the selected node, its partition has to be determined using eq. 12 and 13. This takes $O(d + k)$ time, as we consider only the neighbors of the selected node to compute partition score. Steps 2-4 are repeated for T' iterations. Hence overall running time is $O\big((\alpha \times \beta \times |\mathcal{E}|) + (|\mathcal{V}| \times d + k) \times T'\big)$. Typically α, β and k are $<< |\mathcal{V}|$ and $T' = o(|\mathcal{V}|)$. Since $|\mathcal{V}| \times d \approx |\mathcal{E}|$, hence complexity is $o(|\mathcal{E}| \times |\mathcal{V}|)$ Further, for sparse graphs $|\mathcal{E}| = O(|\mathcal{V}|)$. Hence time complexity of NeuroCUT is $o(|\mathcal{V}|^2)$.

A.5 Comparison with non-neural clustering baselines

We compare the performance of our method against graph based clustering algorithms hMETIS and Spectral Clustering and partition algorithm Gomory-Hu Tree [11]. Since these methods are not compatible with datasets having raw node features, we compare with two datasets namely Facebook [22] and Stochastic Block Model(SBM) [1] which don't have node features for this comparision. Table 8 compares the performance of NeuroCUT against hMETIS and Spectral Clustering. The details of these datasets are presented in Table 7. In addition to non-neural methods, we also show the performance of neural methods MinCutPool, DMon, Ortho and GAP on these datasets.

We observe that performance of NeuroCUT is better than non-neural methods hMETIS, Spectral and Gomory-Hu Tree in most of the cases on Facebook dataset. Further, on SBM dataset, it matches the performance of hMETIS and Spectral clustering. We would also like to highlight that in the case of $k - mincut$, Gomory-Hu Tree algorithm generated trivial partitions for these datasets where almost all nodes were assigned the same partition. The other neural methods such as MinCutPool, DMon, Ortho, and GAP fail to produce a valid solution in most of the cases. This is possibly because these baselines are not robust to perform on datasets without raw node features. Overall, we see that NeuroCUT outperforms the baselines on diverse objectives and datasets.

Table 8: Performance of different methods on Facebook and SBM dataset. Lower values are better. '–' denotes *nan*.

Dataset	Method	Metrics			
		Normalized	Sparsest	Balanced	k-MinCut
Facebook	**NeuroCUT**	**0.257**	**4.121**	**0.972**	0.015
	hMETIS	1.15	52.319	1.115	0.200
	Spectral	1.67	4.72	2.459	0.003
	Gomory-Hu Tree	4.00	5.00	4.792	**~0**
	MinCutPool	-	-	-	0.023
	DMon	-	-	-	**~0**
	Ortho	-	-	-	0.05
	GAP	-	-	-	0.025
SBM	**NeuroCUT**	0.191	3.939	0.191	0.038
	hMETIS	0.191	3.939	0.191	0.038
	Spectral	0.191	3.939	0.191	0.038
	Gomory-Hu Tree	4.003	48.75	4.787	0.0075
	MinCutPool	-	-	-	0.0
	DMon	-	-	-	0.0
	Ortho	-	-	-	~0
	GAP	-	-	-	0.035

Dataset	k-Means	Random	DBSCAN
Cora	**0.33**	0.79	0.55
CiteSeer	**0.20**	0.51	0.70

Table 9: Impact of Clustering initialization in Warm-up Phase of NeuroCUT in the normalized cut objective at $k = 5$.

A.6 Impact of Clustering initialization in Warm Start Phase of NeuroCUT

To understand the importance of different initializations in the warm-up phase, we perform an ablation study on 2 datasets namely Cora and CiteSeer using 3 different initialization namely *k-Means(default)*, density-based clustering *DBSCAN*, [9] and Random initialization. In table 9 we observe the performance on normalized cut at $k = 5$. We observe that *k-Means* performs the best in this experiment. The improvement observed when using *k-Means* or *DBSCAN* over Random shows that a good initialization i.e. warm-up does help in improving quality of partitions. For this experiment, in DBSCAN we set $eps = 0.9$ and $min\_samples = 2$. It's worth noting that DBSCAN's performance can vary based on parameter selection. Nonetheless, the primary goal of this experiment is to demonstrate the advantageous role of a good initialization, specifically in contrast to random initialization.

A.7 Impact of Cluster Strength on Performance

To understand the impact of community structure on performance of NeuroCUT, we generate Stochastic Model Block(SBM) graphs with different intra cluster strength. In Table 10, we observe the performance of different methods on the normalized cut objective at $k = 5$. The baseline methods hMETIS and Spectral Clustering perform worse when the strength within communities is low indicated by Intra Cluster Edge Probability and Clustering Coefficient values. Although NeuroCUT outperforms or matches existing methods in all cases, however, the gap between baselines and NeuroCUT increases significantly when the community structure is not strong. Further, the neural baselines DMon, MinCutPool and Ortho generated 'nan' values for this experiment.

Table 10: Performance on SBM Dataset against varying community strength.

	Dataset Statistics				Normalized Cut		
SBM	Intra Cluster Edge Probability	Clustering Coefficient	$\|V\|$	$\|E\|$	NeuroCUT	hMETIS	Spectral Clustering
1	0.005	0.003	495	555	**0.3089**	0.331	0.379
2	0.01	0.0029	525	661	**0.3434**	0.5889	0.4259
3	0.2	0.184	500	5150	0.1912	0.1912	0.1912
4	0.4	0.3822	500	10102	0.1153	0.1153	0.1153

A.8 Impact of β on NeuroCUT

In this section, we study the impact of β parameter(eq. 6) which is the number of iterations in random walk with restart. In Table 11 we present the performance of NeuroCUT using different β on normalized-cut objective at $k = 5$. We observe that NeuroCUT improves with more iterations as β increases and then its performance stabilizes.

β	Normalized Cut
1	2.12
3	1.57
10	0.33
50	0.33
100	0.33

Table 11: Impact of β parameter on NeuroCUT on normalized cut at $k = 5$ on the Cora dataset.

A.9 Time complexity of different methods

Table 12 presents the complexities of NeuroCUT and other prominent neural and non-neural baselines algorithms. While some neural algorithms exhibit faster complexity, they require separate training for each partition size (k). In contrast, NeuroCUT, once trained, can generalize to any value. Consequently, for practical workloads, NeuroCUT presents a more scalable option in terms of computation overhead and storage (one model versus separate models for each). In the table, $|\mathcal{V}|$ is the number of nodes in the graph, $|\mathcal{E}|$ refers to number of edges, k refers to the number of partitions and d refers to the average node degree.

Method	Complexity	Comments
DMon	$O(d^2 \times \|\mathcal{V}\| + \|\mathcal{E}\|)$	Per iteration complexity
MinCutPool	$O(d^2 \times \|\mathcal{V}\| + \|\mathcal{E}\|)$	Per iteration complexity
Ortho	$O(\|\mathcal{V}\| \times k^2)$	Per iteration complexity
GAP	$O(\|\mathcal{V}\| \times k^2 + \|\mathcal{V}\|^2)$	Per iteration complexity
Spectral	$O(\|\mathcal{V}\|^3)$	Inference complexity
NeuroCUT	$o(\|\mathcal{V}\|^2)$	Inference complexity

Table 12: Time complexity of different methods.

The quadratic time complexity of NeuroCUT may pose challenges for very large graphs. However, this complexity remains faster than spectral clustering, a widely used graph partitioning algorithm. Also, for the baseline neural methods(DMon, MinCutPool, Ortho and GAP), the time complexity provided is per iteration. The number of iterations often ranges between 1000 to 2000 and there is no clear understanding of how it varies as a function of the graph (like density, diameter, etc.)