A Machine Learning Heuristic for Streaming Graph Partitioning

Master's Thesis

to obtain the academic degree of

Master of Science (M.Sc.)

in Applied Computer Science

at Algorithm Engineering Group Heidelberg

Karls-Ruprecht University Heidelberg

by

Felix Roland Hausberger

Supervisor

Univ.-Prof. PD. Dr. rer. nat. Christian Schulz

Co-Referee

Bora Uçar, Senior Researcher DR2, CNRS and ENS de Lyon

Co-Referee

Marcelo Fonseca Faraj, Doctoral Student

Matriculation number: 3661293

Date of submission: 19. September 2022

Editing period: 18.03.2022 - 19.09.2022

Dedicated to Casa Montana

Declaration in Lieu of an Oath

I hereby certify that I have written my Master's thesis on the topic

A Machine Learning Heuristic for Streaming Graph Partitioning

independently and have used only the sources and aids indicated and that the principles and recommendations "Responsibility in science" of the University of Heidelberg have been considered. The work has not yet been submitted to any other examination authority and has not been published. I also certify that the electronic version submitted is the same as the printed version.

Heidelberg, 19. September 2022

F. Hausberger

Hausberger, Felix Roland

Acknowledgements

First and foremost, I am deeply thankful to both my supervisors Prof. Dr. Christian Schulz from University Heidelberg and Prof. Dr. Bora Uçar from ENS Lyon for the guidance, mentoring, and support while doing my Master's thesis and for giving me the possibility to defend my thesis at Lyon. Also, I want to thank Marcelo Fonseca Faraj from Christian's Algorithm Engineering group for his open ear in case I faced challenges during the project's implementation. Last but not least, I would like to thank my family, friends, and colleagues from SAP for their continuous support.

Abstract

- English -

In scientific computing a common problem is to schedule workloads onto processors of parallel machines, that should have an almost equal amount of load but also the ability to work close to independently from each other with as little communication as possible. In theoretical computer science, this problem is described as the NP-complete graph partitioning problem [1, 2]. In the case the graph to be partitioned does not fit into the main memory of a machine or is simply evolving online over time, streaming graph partitioning algorithms are necessary. While multiple algorithmic heuristics were introduced in the past for such streaming algorithms [3, 4, 5, 6, 7, 8, 9], applicable machine learning based heuristics are still very recent [10, 11, 12, 13].

In this work, we investigate both conventional machine learning models as well as deep learning models in their capability to serve as a heuristic in a streaming graph partitioning algorithm. We especially focus on supervised machine learning techniques based on block labels computed by Karlsruhe Fast Flow Partitioner (KaFFPa), a sophisticated offline graph partitioner from Sanders and Schulz [14]. Our novel buffered streaming model supports both node and edge streams. We incorporate the models into a framework to partition non-attributed node streams and study different features from statistical, greedy, and heuristic feature groups. As the streamed graph is enriched with features online, the algorithm is applicable to any abstract graph without available features from an application domain as, to the best of our knowledge, the very first approach in machine learning based streaming graph partitioning literature. Also, we introduce a novel prediction propagation concept to improve on predictions made inside the streaming buffer. In the end, we are able to outperform our main competitor GCNSplit [12, 13] by 6.11% to 22.29% measured by the replication factor in an edge streaming scenario.

Abstract

- Deutsch -

Im wissenschaftlichen Rechnen besteht ein häufiges Problem darin, Arbeitslasten auf Prozessoren paralleler Maschinen aufzuteilen, die nahezu gleiche Last haben sollten, aber auch die Fähigkeit mit möglichst wenig Kommunikation nahezu unabhängig voneinander zu arbeiten. In der theoretischen Informatik wird dieses Problem als NP-vollständiges Graphpartitionierungsproblem beschrieben [1, 2]. Wenn der zu partitionierende Graph nicht in den Hauptspeicher einer Maschine passt oder sich einfach online über die Zeit entwickelt, sind Algorithmen zur Stream-basierten Partitionierung von Graphen erforderlich. Während in der Vergangenheit mehrere algorithmische Heuristiken für solche Stream-basierten Algorithmen vorgestellt wurden [3, 4, 5, 6, 7, 8, 9], sind anwendbare, auf maschinellem Lernen basierende Heuristiken noch sehr jung [10, 11, 12, 13].
In dieser Arbeit untersuchen wir sowohl konventionelle maschinelle Lernmodelle als auch Deep Learning Modelle auf ihre Fähigkeit, als Heuristik in einem Stream-basierten Graph-Partitionierungsalgorithmus zu fungieren. Wir konzentrieren uns insbesondere auf überwachte maschinelle Lerntechniken, die auf den von KaFFPa errechneten Blockzuweisungen basieren, einem hochentwickelten Offline-Graphpartitionierer von Sanders und Schulz [14]. Unser neuartiges Puffer-basiertes Streamingmodell unterstützt sowohl Knoten- als auch Kantenströme. Wir integrieren die Modelle in ein Framework, um nicht-attributierte Knotenströme zu partitionieren und studieren verschiedene Features aus statistischen, gierigen und heuristischen Featuregruppen. Da der gestreamte Graph online mit Features angereichert wird, ist der Algorithmus auf jeden abstrakten Graphen ohne verfügbare Features aus einer Anwendungsdomäne anwendbar, was unseres Wissens nach der erste Ansatz in der Literatur Stream-basierter Partitionierung von Graphen mit Hilfe von maschinellem Lernen ist. Außerdem führen wir ein neuartiges Konzept zum Propagieren von Vorhersagen ein, um die Vorhersagen innerhalb des Streaming-Puffers zu verbessern. Am Ende sind wir in der Lage, unseren Hauptkonkurrenten GCNSplit [12, 13] gemessen am Replikationsfaktor in einem Kantenstrom-Szenario um 6,11% bis 22,29% zu übertreffen.

Contents

List of Abbreviations

BFS	Breadth-First Search
CART	Classification and Regression Tree
CNN	Convolutional Neural Network
DIMACS	Center for Discrete Mathematics and Theoretical Computer Science
GAP	Generalizable Approximate Graph Partitioning Framework
GBDT	Gradient Boosted Decision Tree
GCN	Graph Convolutional Network
HDRF	High Degree Replicated First
KaFFPa	Karlsruhe Fast Flow Partitioner
LDG	Linear Deterministic Greedy
RBF	Radial Basis Function
PCA	Principal Component Analysis
SNAP	Stanford Network Analysis Project
SVM	Support Vector Machine
t-SNE	t-Distributed Stochastic Neighbor Embedding
VLSI	Very Large-Scale Integration

CHAPTER 1

Introduction

1.1 Motivation

As a fundamental data structure in computer science, graphs can serve as an abstraction for many real-world scenarios dealing with entities and their relation to each other. In many applications, it is necessary to partition such a graph into a fixed number of blocks, such that all blocks comply with a balance constraint while having as few edges cut as possible. This ensures an application's ability to scale in quality, time, or costs by reducing complexity or enabling parallelization.

The most common example in the context of graph partitioning lies in the domain of scientific computing when trying to schedule workloads onto processors of parallel machines, that should have an almost equal amount of load but also the ability to work close to independently from each other with as little communication as possible [15]. In Very Large-Scale Integration (VLSI) system design, graph partitioning is used to subdivide a conglomerate of millions of transistors into smaller, more manageable components while keeping the number of electrical signals to be exchanged low [16]. More interesting in the domain of software engineering is query optimization in distributed databases, where data often queried together shall be assigned to the same shard to reduce query processing time [17]. Application domains of (hyper)graph partitioning reach even further beyond up to image segmentation, quantum circuit simulation, complex network analysis tasks, and more [15, 17].

While having a lot of application domains in practice, the graph partitioning problem is also interesting from a theoretical computer science point of view. The problem is NP-complete [1, 2] and the perfectly balanced version of the problem has no finite-factor approximation algorithm [18], which is why heuristics are often used in practice to address the problem [15]. The graph partitioning problem can be solved on different computational models with each of them having its own arguments to be used [17]. If the graph fits into the main memory and is known in advance, most often sequential algorithms are used that can be designed to work either completely internal memory or shared memory parallel. Both of them produce high-quality partitions with the shared memory version requiring less time, but more computing resources. Besides the sequential algorithms also distributed memory parallel algorithms exist, that scale well to large graph instances but typically give lower partitioning quality. They are often used when trying to partition huge graphs, even on cheap machines, yet these algorithms require pre-splitting the graph to be partitioned before running the actual graph partitioning algorithm itself. All of those algorithms can in theory be run inside a memetic framework to receive even higher quality partitions, in practice this only works for smaller graphs due to its high running time efforts and resource requirements. The best solution is of course produced by an exact but long-running combinatorial algorithm, which does not scale to large graph instances due to its theoretical complexity.

If a graph does not fit into memory or graph partitioning should be performed online, streaming graph partitioning algorithms are used. They are fast and require little memory, the quality of the partitions they produce is nevertheless inferior to those of offline algorithms. Algorithmic approaches to do streaming graph partitioning include the one-pass streaming approaches Linear Deterministic Greedy (LDG) [3], Fennel [4] and AKIN [5]. Nishimura and Ugander [6] proposed ReLDG and ReFennel, an adaptation of the original LDG and Fennel heuristics that can be applied when a graph is streamed consecutively multiple times. Last but not least, there are buffered streaming graph partitioning approaches like WStream [8] and HeiStream [9] that do not stream the graph node by node but use a streaming buffer in between to have a more global view at hand when doing block assignments. Streaming algorithms have even been used in offline partitioning scenarios as Jafari et al. [19] showed with a shared memory multilevel algorithm based on LDG.

Besides these algorithmic approaches, there are a few ideas to utilize machine learning techniques for graph partitioning. For offline graph partitioning the most known approach is the Generalizable Approximate Graph Partitioning Framework (GAP) introduced by Nazi et al. [10], which inspired other approaches like the one from Gatti et al. [11] that enriches GAP with spectral graph partitioning theory. For streaming graph partitioning, there is only one approach using machine learning called GCNSplit from Abbas [12] and Zwolak et al. [13], which was also inspired by GAP [10]. These machine learning based approaches have in common to work only with *attributed* graphs, i.e., with node attributes from a specific application domain. Furthermore, they are based on unsupervised learning optimizing custom loss functions. GCNSplit also focuses on edge partitioning only. The benefits of such models lie in the fact that the long-lasting and intricate process of designing algorithmic heuristics manually can be replaced by learning partitioning patterns and correlations from the data.

1.2 Contribution

In this thesis, we investigate both conventional machine learning models as well as deep learning models in their capability to serve as a heuristic in a streaming graph partitioning algorithm. The models to be evaluated include the conventional classification models Logistic Regression, Support Vector Machine (SVM), Gradient Boosted Decision Trees (GBDTs) and the deep learning models GraphSAGE and a custom Partitioner model that applies an additional densely connected network on top of GraphSAGE. We especially focus on supervised machine learning techniques based on block labels computed by KaFFPa, a sophisticated offline graph partitioner from Sanders and Schulz [14]. Our novel buffered streaming model supports both node and edge streams. We incorporate the models into a framework to partition non-attributed node streams and study different features from statistical, greedy, and heuristic feature groups. As the streamed graph is enriched with features online, the algorithm is applicable to any abstract graph without available features from an application domain. As far as we know, this is the first attempt in machine learning based streaming graph partitioning literature at addressing this use case. By generating features based on Fennel scores, we even combine conventional approaches to address streaming graph partitioning for feature generation with modern machine learning methodologies during prediction. Using our newly introduced concept

of prediction propagation, we are able to add a crucial ingredient when utilizing Fennel score based features to improve on predictions made inside the streaming buffer.

Almost all of our models are able to outperform the Fennel based baseline algorithm by partitioning quality in evolving graph streaming scenarios, where nodes are added over time to the graph. For graphs that originate from the same generative model, the best performing model based on XGBoost's implementation of GBDTs [20] is even able to predict partitions on before totally unseen graphs decently well. Hard balance constraint heuristics make sure the resulting partitions are balanced every time the graph is streamed. In the end, we are able to outperform our main competitor GCNSplit by 6.11% to 22.29% measured by the replication factor in an edge streaming scenario.

1.3 Thesis Structure

In the beginning in Chapter 2, all necessary fundamentals for the framework introduced in this thesis are examined. Starting with the definition of streaming graph partitioning, conventional algorithmic heuristics to address the problem, and models that are being used throughout the thesis including Logistic Regression, GBDTs, SVM, and GraphSAGE.

Next, we cover related work in Chapter 3. We give an overview of graph partitioning, conventional algorithms for streaming graph partitioning, followed by offline, deep learning based graph partitioning algorithms and a true streaming graph partitioning deep neural network.

The Chapter 4 introduces the streaming model, the grouping of considered features for non-attributed graphs, and a new concept called prediction propagation that increases partitioning quality for buffered streaming settings. Furthermore, the hard balance heuristics are explained as well as the architectural decisions for each model.

The biggest part of this thesis is formed by the empirical evaluations as given in Chapter 5. Here the two datasets for training/tuning and evaluation are investigated and the initial configurations for the upcoming experiments are set. Then we begin an extensive study of the different features for each model and draw a final feature selection conclusion. Afterwards, we present preprocessing methods to prepare the feature space for better prediction capabilities for some models and tune their individual hyperparameters. Ex-

periments about the streaming configuration include the choice of a balance heuristic, streaming buffer, streaming step sizes, and the number of prediction propagation rounds. A short excursion into a Breadth-First Search (BFS) based streaming order vs. a natural streaming order is also given. After all the tuning is done we explore the generalization capabilities of each model in evolving graph streaming settings and settings that require predictions on totally unseen graphs from the same structural group. Based upon these insights, we compare the different models and make a recommendation on which model to use for which scenario. For this decision also the running times during training and prediction are considered. Last but not least, a comparison to two competitor algorithms, GCNSplit [12, 13] based on a machine learning heuristic, and 2PS-L [21] based on a pure algorithmic heuristic, is made.

A summary and discussion of this work are given in Chapter 6. Section 6.1 concludes all novelties and findings, whereas Section 6.2 discloses further interesting areas for future work.

CHAPTER 2

Fundamentals

2.1 Streaming Graph Partitioning

First, we introduce the NP-complete [1, 2] *graph partitioning* problem formally. The input is a graph G, the number of blocks k the graph should be partitioned into, and an imbalance parameter ϵ. The graph $G = (V, E, c : V \to \mathbb{R}_{\geq 0}, \omega : E \to \mathbb{R}_{>0})$ is undirected and has n nodes in set V, m edges in set E, nonnegative node weights c, and nonnegative edge weights ω without multi-edges or self-loops. Graph partitioning can be performed in two variations.

In *vertex partitioning* we partition G into $k \in \mathbb{N}_{>1}$ blocks $V = V_1 \cup V_2 \cup ... \cup V_k$ such that $\forall i, j : V_i \cap V_j = \varnothing \ (i \neq j)$ and such that the *total cut size* $\sum_{i<j} \omega(E_{ij})$ with $E_{ij} = \{\{u, v\} \in E : u \in V_i, v \in V_j\}$ is minimized while considering a *balance constraint* $\sum_{v \in V_i} c(v) \leq (1 + \epsilon)\frac{\sum_{v \in V} c(v)}{k} = L_{max}, \forall i \in \{1, ..., k\}$ for some imbalance parameter $\epsilon \geq 0$. Note that there exist also other objective functions for vertex partitioning besides the total cut size like the *maximum communication volume* which we will not have a look at in this work.

In the second variant, *edge partitioning*, we partition G into $k \in \mathbb{N}_{>1}$ blocks $E = E_1 \cup E_2 \cup ... \cup E_k$ such that $\forall i, j : E_i \cap E_j = \varnothing \ (i \neq j)$ and such that the *replication factor* $\frac{1}{|V|} \sum_{i \in [k]} V(E_i)$ with $V(E_i) = \{u \in V : \exists \{u, v\} \in E_i \cup \exists \{v, u\} \in E_i\}$ is minimized while considering a *balance constraint* $\sum_{\{u,v\} \in E_i} \omega(\{u, v\}) \leq (1 + \epsilon)\frac{\sum_{\{u,v\} \in E} \omega(\{u,v\})}{k} = L_{max}, \forall i \in \{1, ..., k\}$ for some imbalance parameter $\epsilon \geq 0$.

Conventional graph partitioning expects the input graph instance to be available offline and in memory while *streaming graph partitioning* describes a setting, where either the entire graph instance is not known before and evolves online over time by adding more and more nodes from a stream of nodes, or the entire graph is too huge to be processed by an internal memory algorithm, which only has $O(n)$ space available, and needs to be streamed from the disk. In either case not having all global information available for graph partitioning makes it much harder to compete with the in-memory scenario. Often a buffer as big as the available internal memory is used during streaming to obtain a more global view over the nodes or edges to be partitioned and therefore better partitioning qualities. In the case of a true online algorithm, this buffer might come with the downside of delayed block assignment decisions for individual nodes or edges depending on the streaming model.

2.2 Algorithmic Heuristics

Pure algorithmic heuristics for streaming graph partitioning used in this work are *LDG* by Stanton and Kliot [3] and *Fennel* by Tsourakakis et al. [4]. In both heuristics, a node v is assigned to block idx with the highest score. LDG is defined as

$$idx = \arg\max_{i \in [k]} \{|V_i \cap N(v)| \cdot (1 - \frac{|V_i|}{L_{max}})\}$$

whereas Fennel can be formulated as

$$idx = \arg\max_{i \in [k]} \{|V_i \cap N(v)| - \alpha\gamma|V_i|^{\gamma-1}\}$$

with α and γ chosen as $\alpha = \sqrt{k}\frac{m}{n^{3/2}}$ and $\gamma = 1.5$ as evaluated in the original paper. Both heuristics try to minimize the total cut size while punishing imbalance, but both heuristics are only suitable for bounded node streams as the number of nodes n needs to be known in advance. The heuristics differ in the term to punish imbalance, which is a linearly decreasing multiplicative for LDG and an exponentially growing subtraction term for Fennel.

2.3 Logistic Regression

The following explanation is based on a machine learning foundations book by Géron [22]. *Logistic Regression* is the most simple binary classification model and is defined as

$$p(x_i;\theta) = \frac{1}{1+e^{-\theta^T x_i}}$$

with x_i being the feature vector of instance i, θ being the trainable model parameters and $p(x_i,\theta)$ being the probability that instance i belongs to the class. It assumes the data to be linearly separable and learns the model parameters θ by minimizing the negative log-likelihood function

$$Loss(x_i,y_i;\theta) = -(y_i \log(p(x_i,\theta)) + (1-y_i)\log(1-p(x_i,\theta))).$$

When trying to predict multiple categories one can either train the model in a one vs. rest fashion or use the multinomial Logistic Regression. It replaces the sigmoid function $S(x) = \frac{1}{1+e^{-x}}$ with the softmax function to give the probability distribution over all classes instead of just a single probability value and is defined as

$$p_c(x_i;\theta_c) = \frac{e^{-\theta_c^T x_i}}{\sum_{i=1}^{C} e^{-\theta_i^T x_i}}.$$

2.4 Gradient Boosted Decision Trees

The following explanation of the *decision tree* is based on the machine learning foundations book by Géron [22]. A decision tree is a simple, explainable machine learning model that can be used both for regression and classification. It tries to split a dataset on every node into smaller subsets such that the final leaves have low *impurity* meaning each subset should best only contain instances of the same class (classification) or continuous values with small variance (regression). To do this, the Classification and Regression Tree (CART) algorithm is applied, a greedy algorithm that goes over every feature k and threshold t_k, and picks (k,t_k) such that the decrease in impurity is maximized on

the current node. In other words, the goal is to greedily minimize the variance of the instances in each subset. The final prediction in each leaf is either the class priors for classification or the average of the target values in case of regression. In classification, the *Gini-Impurity* metric $G_i = 1 - \sum_{c=1}^{C} p_{i,c}^2$ is often used to measure the impurity score at node i with $p_{i,c}$ being the class prior of class c within node i. In regression, the mean squared error is commonly used as a measure of impurity. Note that finding the optimal CART is NP-complete. Decision trees are parameter-free models, which is why they need regularization to prevent their natural tendency for overfitting. Such regularization could mean fixing the tree depth, the minimum and the maximum number of leaves, or the number of nodes at least needed for a split.

Boosting is a technique that constructs an ensemble of weak learners that are connected to an additive meta-model. The weak learners, often decision trees or decision stumps (decision trees of depth one), are connected sequentially such that one estimator improves on the previous model or "boosts" the previous model. The weak learners are called weak, as they could only make poor predictions when being on their own. *Gradient boosting* is a subtype of boosting and boosts previous models in the chain by predicting the residual error of the previous model. Besides gradient boosting there is also *AdaBoost*, which improves on previous models by giving wrongly predicted data instances a higher weight when fitting the next weak learner [22].

To understand the principles of *Gradient Boosted Decision Trees (GBDTs)* we first want to explain it in the context of regression before coming to the classification domain. Note that in regression we try to predict a function with the target values y given a feature matrix X.

Algorithm 1 Gradient Boosted Decision Trees learning procedure [23]

Let $F_0(X) = \frac{1}{N} \sum_{i=1}^{N} y_i$ be the mean of target y across all instances i
for $m = 1 \dots M$ **do**
    Let $r_{m-1} = y - F_{m-1}(X)$ be the residual direction vector
    Train regression tree Δ_m on r_{m-1}, minimizing squared error
    $F_m(X) = F_{m-1}(X) + \eta \Delta_m(X)$
return F_M

Algorithm 1 shows the gradient boosting approach for M weak learners and a learning rate of η, which determines the influence of each estimator on the final prediction. The lower the learning rate η, the more predictors are needed to fit the function, but the

less the ensemble is likely to overfit. This regularization technique is called *shrinkage*. Another regularization technique is subsampling the data before learning a weak classifier on it. If single estimators are only trained on 60% of the available training data, this reduced the variance but may increase the bias of the final prediction. The variance-bias trade-off is a typical problem in machine learning when it comes to regularization [22].

The reason for the name *gradient* boosting lies in the residual r, which is the negative of the gradient of the mean squared error loss function, that we try to optimize. Generally, we can use this gradient boosting approach to optimize any loss function of choice as long as the weak learners are trained to predict the negative of the loss function's gradient [23].

Coming from a regression task over to a classification task, the question appears, how to predict categorical values if gradient boosting seems to be only useful to approximate functions in a regression context? To answer this question, we look at the binary classification case, which is still easy to explain. Here GBDTs predict the logarithm of the odds across the feature space with the odds being defined as $\frac{P(Y=1)}{P(Y=0)}$. The logarithm makes the ratio symmetric around zero. While predicting the logarithm of the odds, the decision trees actually fit the residual probabilities that lie between zero and one. We can transform the logarithm of the odds anytime back to prediction probabilities using the following function:

$$P(Y = 1) = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

This means we get predictions for all data instances in the form of the logarithm of the odds γ. We then transform these predictions to probabilities using the above formula and calculate the residuals r_i (observed probability - predicted probability) for every data instance i. Subsequently, we fit the next weak learner to those probability residuals. For each leaf $L_l = \{r_1, \ldots, r_n\}$ in model t, we calculate the prediction to be output using the following formula:

$$\gamma_{l,t} = \frac{\sum_{i \in L_l} r_i}{\sum_{i \in L_l} p_{i,t-1}(1 - p_{i,t-1})}$$

Using the incremental inference formula, we calculate the new prediction up to estimator M as

$$\gamma = \gamma_0 + \eta\gamma_1 + \cdots + \eta\gamma_M .$$

Then the process is repeated until all weak learners are fit. Interestingly, the residual is the negative gradient of the negative log-likelihood loss function often used for classification [24].

For a multi-class classification scenario, such a model can always be trained in a one vs. rest fashion. Different implementations exist for GBDTs with the most common ones being XGBoost [20], LightGBM [25] and CatBoost [26].

2.5 Support Vector Machine

The *Support Vector Machine (SVM)* algorithm is a well-known classifier used in supervised machine learning. A comprehensive survey on SVM classification was published by Cervantes et al. [27]. In binary classification problems, where the data can be separated linearly, the SVM constructs a decision hyperplane with maximum margin to the support vectors x^+ and x^-, which are the data points closest to the decision hyperplane. Maximizing the margin to the support vectors also means maximizing the generalization capabilities of the model. For a two-dimensional feature space, this behavior is drawn in Figure 2.1.

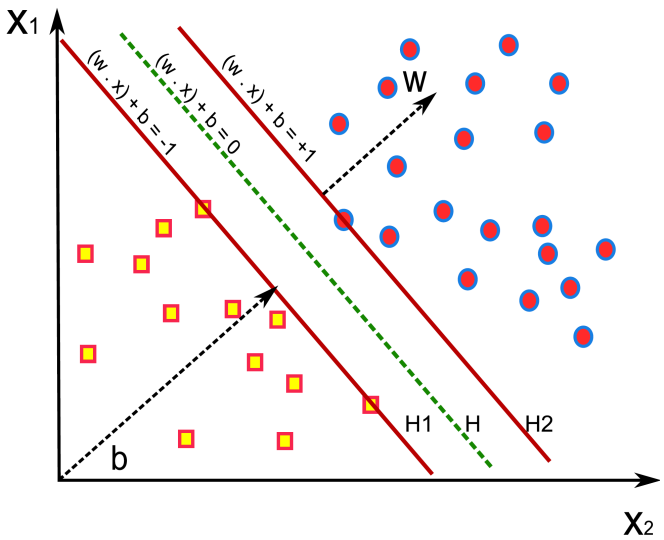


Figure 2.1: Linear Support Vector Machine [27]

Let γ be the margin of the decision hyperplane to the support vectors, w be the normal of the decision hyperplane, x_i be the feature vector of node i and $y_i \in \{-1, 1\}$ be the class label of node i. Fixing the support vectors to have a unit distance to the decision hyperplane means $w^T x^+ + b = 1$ and $w^T x^- + b = -1$. The margin can then be formulated as

$$\begin{aligned}\gamma &= \frac{1}{2}\Big[\big((\frac{w}{\|w\|})^T x^+\big) - \big((\frac{w}{\|w\|})^T x^-\big)\Big] \\ &= \frac{1}{2\|w\|}\big((w^T x^+) - (w^T x^-)\big) \\ &= \frac{1}{\|w\|}\end{aligned}$$

giving us the optimization problem in its primal form as

$$\begin{aligned}&\min_{w,b} \|w\|^2 \\ s.t.: \quad & y_i(w^T x_i + b) \geq 1 \quad \forall i.\end{aligned}$$

Often the given data cannot be separated linearly. In this case, one could either make use of a soft margin hyperplane in case there is a linear separation tendency or one needs to map the data into a higher dimensional feature space using a mapping function $\phi(x)$, in which the data is linearly separable again. As the latter might lead to high computational costs, one makes use of the *kernel trick* on the SVM's dual optimization problem formulation to learn the decision hyperplane in the higher-dimensional feature space, without ever having to transform the original data practically.

Using the representer theorem [28], w can always be formulated as a linear combination of the original features, i.e., $w = \sum_j \alpha_j y_j x_j$. Inserting this into the primal formulation gives the dual optimization problem formulation:

$$\min_{\alpha_j} \sum_{jk} \alpha_j \alpha_k y_j y_k (x_j^T x_k)$$
$$s.t.: \quad y_i(\sum_j \alpha_j y_j (x_j^T x_i) + b) \geq 1 \quad \forall i.$$

In the original constraint $y_i(\sum_j \alpha_j y_j (x_j^T x_i) + b) \geq 1$ the input feature vectors would be mapped to the higher dimension as $y_i(\sum_j \alpha_j y_j \langle \phi(x_j), \phi(x_i) \rangle + b) \geq 1$. The scalar product $\langle \phi(x_j), \phi(x_i) \rangle$ can now be replaced by a *kernel function* $K(x_i, x_j)$ that computes the same result, but is far less computationally expensive than doing it the normal way. Using this kernel trick, one can learn decision hyperplanes for non-linear separable data without having to map the data into a higher dimensional space.

Kernel functions need to fulfill specific properties that we will not go into detail about in this work. Commonly used kernel functions include the polynomial kernel function $K(x_i, x_j) = (x_i^T x_j + 1)^p$ of degree p and the Radial Basis Function (RBF) kernel $K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$ with γ being the inverse of the radius of influence of support vector i . Which kernel is considered to be best depends on the actual data and needs to be explored empirically. Subsequently, the kernel-specific hyperparameters need to be tuned. Also, regularization is needed to not overfit the data in the theoretical higher dimensional feature space.

Besides the inherent algorithmic complexity also the choice of data to be used for training is important. Computing the kernel function for all training data pairs leads to the kernel matrix and has a quadratic effort. This is why the training data set should not be chosen too big when utilizing SVMs. Actually, only such data is important that will form the support vectors and contributes to the formation of the decision hyperplane during optimization. On the other hand, there are multilevel approaches to train SVMs faster like the one from Schlag et al. [29]. Furthermore, the training data set should be balanced as otherwise an unintended bias towards a specific class might be included in the model.

Last but not least, a decision hyperplane can only differentiate between two classes. In a multi-class setting, a one vs. one or a one vs. rest competition needs to be performed increasing training and prediction time.

In the end, the SVM is still a powerful classifier nowadays and is often used as an alternative to neural network based approaches that are difficult to engineer as having a lot more parameters to tune regarding their architecture and the learning algorithm used. Often neural networks are also sensitive to noise in the data, data normalization, and the weight initialization of the network itself making the training effort even larger.

2.6 GraphSAGE

While previous transductive approaches to generate low-dimensional node embeddings like DeepWalk [30] or node2vec [31] do not generalize to unseen nodes during prediction, *GraphSAGE* [32] is an inductive node embedding framework that promises to generate feasible embeddings in a setting of evolving graphs or unseen graphs with identical node features from a similar application domain. The node embeddings serve as features for downstream node prediction and graph analysis tasks. They should not just encode individual node information, but also structural and semantic information about the local neighborhood of each node. The original high-dimensional node features can be of application domain-specific nature or conventional node characteristics like the degree of a node. So instead of learning concrete hard-wired node embeddings as transductive methods do, GraphSAGE learns a set of *aggregator functions* that aggregate node features of a local neighborhood for each node to create corresponding embeddings generically while also applying dimensionality reduction.

Algorithm 2 GraphSAGE forward propagation [32]

procedure FORWARD($\mathbf{x}_v, K$)
  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v,\ \forall v \in V$
  for $k = 1 \dots K$ **do**
    for $v \in V$ **do**
      $\mathbf{h}_{N(v)}^k \leftarrow Aggregate_k(\{\mathbf{h}_u^{k-1},\ \forall u \in N(v)\})$
      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot Concat(\mathbf{h}_v^{k-1}, \mathbf{h}_{N(v)}^k))$
    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2,\ \forall v \in V$
  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K,\ \forall v \in V$
  return $\mathbf{z}_v$

Algorithm 2 describes how the node embeddings are generated using the GraphSAGE framework. Starting from individual node features $\mathbf{x}$, K aggregation rounds are performed

around the local neighborhood of every node resulting in temporary embedding states $\mathbf{h}^k$ in round k, that are then being normalized to a unit length embedding vector. To calculate the temporary embedding state of a node, the temporary embedding states of a uniformly sampled fixed-size subset of the node's neighbors are aggregated by an k-dependent aggregation function learned during training. The aggregated temporary embedding states are then concatenated with the node's current embedding state and transformed by a dense layer, which results in the node's new temporary embedding state. After K rounds, which represent the range of the local neighborhood in hops to be encoded, the final node embeddings $\mathbf{z}$ are all generated.

As an aggregation function different architectures invariant to the order of a node's neighborhood were evaluated, among them:

1. Mean aggregator: $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot Mean(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1},\ \forall u \in N(v)\}))$
2. LSTM aggregator
3. Max/Mean pooling aggregator: $Aggregate_k^{pool} = Max(\{\sigma(\mathbf{W}\mathbf{h}_u^k + \mathbf{b}),\ \forall u \in N(v)\})$

The mean aggregator applies an element-wise mean over the components of each vector $\mathbf{h}^k$ in the direct local neighborhood of a node including the node itself and skips the neighborhood aggregation step from the base algorithm. LSTMs have more degrees of freedom and therefore possibly better expressive capabilities. The max/mean pooling aggregator first transforms each temporary embedding state by a dense layer (whereas also more complex transformations are possible), and then applies an element-wise max or mean over the components.

Training is possible both in a supervised or unsupervised manner using stochastic gradient descent or a mini-batch approach. Standard cross-entropy loss is being used for supervised training, while for unsupervised training the following negative log-likelihood loss function is to be minimized:

$$Loss(z_v) = -\log(\sigma(\mathbf{z}_v^T \mathbf{z}_u)) - Q \cdot \mathbb{E}_{u_n \sim P_n(u)} \log(\sigma(-\mathbf{z}_v^T \mathbf{z}_{u_n})).$$

When calculating the loss for the embedding of node v, we sample a close neighbor u and expect their embeddings $\mathbf{z}$ to be similar. Therefore the scalar product $\mathbf{z}_v^T \mathbf{z}_u$ would push the log-probability to zero. The same holds for the second term in the loss function,

where Q negative samples from $P_n(u)$ are drawn whose embeddings should by expectation deviate stronger from the embedding of node v.

Experiments on a citation graph derived from Thomson Reuters Web of Science Core Collection and a web graph based on Reddit posts showed that GraphSAGE has a 13.8% (citation graph) and 29.1% (web graph) improvement measured by F1 scores using unsupervised training over a standard Logistic Regression classifier based on raw features combined with embeddings produced by DeepWalk [30]. In a supervised setting, improvements were even higher with 19.7% (citation graph) and 37.2% (web graph). In both scenarios LSTM- and pooling based aggregators scored best [32].

CHAPTER 3

Related Work

3.1 Graph Partitioning using Conventional Methods

Approaches to solving the graph partitioning problem that have proven themselves in practice are based on the multilevel meta-heuristic with the most commonly used model introduced by Hendrickson and Leland [33]. In a coarsening phase the graph is iteratively reduced while the global structure of the graph is still maintained on each level of the multilevel hierarchy. On the coarsest level, the graph is initially partitioned using an ideally high-quality partitioner. Then the graph gets iteratively uncoarsened while applying local search algorithms at the block boundaries for the refinement of the initial partitioning.

Such a multilevel scheme can also be applied inside a memetic framework [34, 35], where a population of partitions of the graph is evolved over several generations. In each round, fit individuals are selected and combined to an improved offspring. To ensure diversity in the population different mutation operators are used on offsprings before individuals are evicted from the population based on their fitness and their similarity to other individuals.

Furthermore, graph partitioning algorithms can be parallelized to either shared memory parallel or distributed memory parallel algorithms [17]. Shared memory parallel algorithms can produce partitions of the same quality as their sequential counterparts but faster while requiring more resources. Distributed memory algorithms on the other hand scale

well to large graphs but often produce partitions with worse quality as each processor has only a limited, local view of the entire graph.

As conventional graph partitioning algorithms are not the focus of this work, we refer the reader to [15, 17, 36] for more detailed information. Conventional graph partitioning algorithms require the graphs to fit into the available internal memory and to be known in advance. If this is not the case, streaming graph partitioning algorithms are necessary, which shall be given an overview in the following section.

3.2 Streaming Graph Partitioning using Conventional Methods

An interesting survey by Çatalyürek et al. [17], an update of the previous survey by Buluç et al. [15], subdivides the space of known vertex streaming graph partitioning algorithms into one-pass streaming, restreaming, and buffered streaming algorithms, which we want to have a look at in the following. Further studies include the one from Abbas et al. [37], which compares different one-pass streaming algorithms for vertex partitioning and edge partitioning by offline as well as application-specific metrics, and Pacaci and Özsu [38], which also evaluates vertex partitioning and edge partitioning algorithms in the field of offline graph analytics and online graph query workloads.

The group of one-pass streaming models contains the algorithmic heuristics LDG [3] and Fennel [4], that are introduced in Section 2.2. In the experiments of Tsourakakis et al., the reported total cut sizes for Fennel are lower compared to those of LDG. Again, both of the heuristics are not suitable for unbounded streams of nodes as they depend on the number of nodes (and the number of edges for Fennel). A further improvement compared to Fennel is achieved by Zhang et al. and their streaming graph partitioning algorithm AKIN [5], which mainly works for edge streams. The core idea is to assign an edge and both of its endpoints to the block maximizing

$$H_i(u,v) = \alpha \cdot sim_i(u,v) - \beta \cdot pnl(i)$$

with $sim_i(u, v)$ being the Jaccard similarity [39] of the neighbors' block assignments evaluated by block i and $pnl(i)$ being a load penalty expressed by the percentage of allowed nodes already assigned to block i. To calculate the Jaccard similarity score only a limited number of neighbors for each node is used containing only the highest degree neighbors. This way the block assignments of structurally important nodes like hubs are stressed and less important nodes by degree are neglected. As being used for edge streams, nodes might switch blocks throughout the streaming process. Zhang et al. report a reduction in the edge-cut ratio of up to 20% compared to Fennel, while still producing well-balanced partitions.

Restreaming algorithms include ReLDG and ReFennel introduced by Nishimura and Ugander [6], that run the same heuristics as in the one-pass models over multiple stream runs to improve the partitioning quality. This has the benefit that future nodes, that were not yet streamed, already have a proposed block assigned after the first streaming run that the heuristics can make use of. Furthermore, ReFennel continuously increases the importance of the balance penalty over the stream runs to achieve balance. Awadelkarim and Ugander [7] build upon ReLDG and evaluate the algorithm with different static (based on a graph's structure) and dynamic (updated between different stream runs) streaming node orders. Their best-performing node order called ambivalence is based on a node's gain inspired by balanced label propagation and places nodes having more ambivalent block assignment decisions at the end of the stream.

Patwary et al. [8] introduce a very simple, but yet effective sliding-window based streaming partitioning algorithm called WStream that is able to beat LDG measured by edge-cut ratio on the selected graph instances using a greedy partitioning strategy. Far more sophisticated is the approach from Faraj and Schulz [9] that uses a multilevel algorithm on a constructed graph based on a streamed batch of nodes and their connections to existing blocks and optionally future nodes. It uses a size-constraint label propagation coarsening technique, initial partitioning based on a weight-considering Fennel heuristic, and again a size-constraint label propagation local search algorithm for uncoarsening, which uses the weight-adapted Fennel function instead of the normal greedy rule. This algorithm called HeiStream only takes linear partitioning time and promises to outperform Fennel for 75.9% on average for small buffer sizes.

Besides the three considered groups of streaming graph partitioning algorithms, multiple instances of such algorithms can also run simultaneously. Sharing the states between

the partitioners can impose a conflict between partitioning quality and communication overhead. Khamoushi introduces a scalable streaming graph partitioning algorithm for such scenarios [40] based on a buffered streaming model and a novel shared-state mechanism. The author reports an average 23% decrease in partitioning time, while at the same time having 15% less communication load with only 5% bigger memory consumption compared to its competitors.

3.3 Graph Partitioning using Deep Learning

3.3.1 Generalizable Approximate Graph Partitioning Framework

One of the first approaches to applying deep learning methods to conventional graph partitioning is described by Nazi et al. [10]. One of its big benefits is the utilization of graph embedding techniques to be able to learn and adapt to the underlying structure of the graphs. This makes it possible for the model to generalize over different graphs with similar structures during prediction without having seen them directly during training. The Generalizable Approximate Graph Partitioning Framework (GAP) is capable to transfer such learned patterns and principles from smaller training graphs with around 1000 nodes to much larger graphs with up to 27 000 nodes.

The training is done in an unsupervised manner. Nazi et al. introduce a custom differentiable loss function, which is a continuous relaxation of the normalized cut objective

$$Ncut(V_1, V_2, \ldots, V_k) = \sum_{i=1}^{k} \frac{cut(V_i, V \backslash V_i)}{vol(V_i)}$$

with the volume of a block being defined as $vol(V_i) = \sum_{v \in V_i} deg(v)$. To transform the objective into a continuous space, the following steps are done.

First, $cut(V_i, V \backslash V_i)$ can be transformed into the term $\sum_{reduce-sum} Y(1-Y)^T \odot A$ with $Y \in \mathbb{R}^{n \times k}$ and $A \in \mathbb{R}^{n \times n}$. Y_{ik} describes the output of the model, which is the probability of node i to belong to block k, A is the adjacency matrix, and the operator $\odot$ performs an element-wise multiplication (Hadamard product). The *reduce-sum* operation sums up

the elements inside the resulting matrix and gives the final expected total cut size of the prediction.

Next, to formulate $vol(V_i)$ a new variable $\Gamma = Y^T D$ is introduced with $D \in \mathbb{N}_{\geq 0}$ being the row vector of all node degrees. Adding an element-wise division by Γ brings us to the final formulation of the continuous relaxation of the expected normalized cut objective:

$$\mathbb{E}[Ncut(V_1, V_2, \ldots, V_k)] = \sum_{reduce-sum} (Y \oslash \Gamma)(1 - Y)^T \odot A.$$

Nazi et al. enrich this formulation with an additional squared error term to also consider the balance of blocks, which gives the final loss function

$$\mathcal{L} = \sum_{reduce-sum} (Y \oslash \Gamma)(1 - Y)^T \odot A + \sum_{reduce-sum} (\mathbf{1}^T Y - \frac{n}{k})^2$$

with $\mathbf{1}^T Y$ being the expected number of nodes in each block.

The model of GAP consists of a graph embedding module and a concatenated graph partitioning module. As the embedding module Nazi et al. utilize a selection of either a 3-layer Graph Convolutional Network (GCN) inspired by Kipf and Welling [41] or the already introduced GraphSAGE framework by Hamilton et al. [32] using the max pooling aggregator with 5 layers and 512 units. The partitioning module is a simple 3-layer fully-connected network with a hidden dimensionality of 64 followed by a softmax function to output the block assignment probabilities for each node.

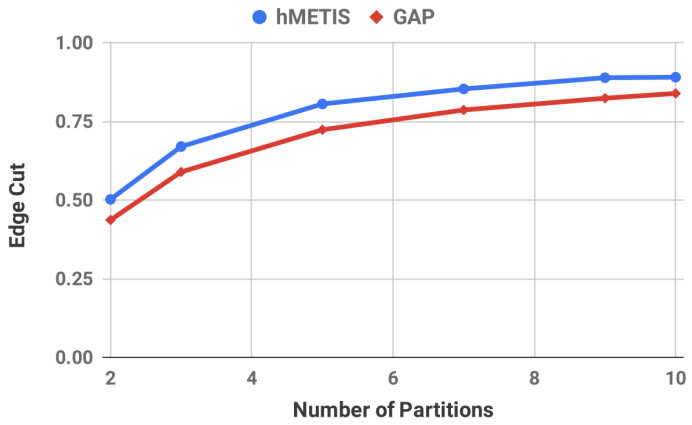


Figure 3.1: Comparison of the fraction of cut edges over k on random graphs between GAP and hMETIS [10]

In a direct comparison for balanced graph partitioning against hMETIS [42], GAP outperforms hMETIS, especially on dense graphs like the artificially generated random graphs as seen in Figure 3.1 provided by Nazi et al. [10]. On sparse real-world TensorFlow computation graphs, the GAP framework matches the performance of hMETIS and even outperforms hMETIS on VGG by 1%. For the real-world graphs, only a 3-partition setting is evaluated, for the random graphs, experiments for $k = 2$ up to $k = 10$ blocks are executed. Both GAP and hMETIS produce 99% balanced partitions in the reported experiments with the balancedness measured as one minus the mean squared error of the number of nodes in every block and a completely balanced block ($\frac{n}{k}$). Note that GAP is optimized for each individual graph to give the best possible result.

When investigating the generalization capability of GAP, one recognizes that generalization works well for unseen graphs with the limitation that the unseen graphs need to be similar to the graphs seen during training. The real-world TensorFlow computation graphs have a very similar degree distribution and so do the generated scale-free and random graphs. Hang et al. come to the conclusion that the Jaccard similarity correlates with lower total cut sizes. For the real-world graphs, training on a graph covering only 1 325 nodes is enough to make decent predictions for a larger graph with 27 114 nodes. There is just a 2% loss in performance compared to optimizing directly on the unseen graph used during prediction. For the generalization experiments on real-world graphs, the TensorFlow operation types are used as the sole node feature leading to a one-hot encoded feature vector of size 1 518. On random graphs, GAP performs almost equally to hMETIS, but is 10-100 times faster. The reason why such a great boost in prediction time can be achieved is not further investigated. As random graphs do not own features, node features are generated by applying Principal Component Analysis (PCA) to the adjacency matrix.

To summarize, GAP gives an initial idea of how to approach graph partitioning with deep learning techniques. It introduces a loss function for graph partitioning and a model architecture that is capable to generalize to unseen graphs to a limited degree.

What remains in order to be applicable for the problem setting of this work is to apply such a deep learning model to streamed, non-attributed graphs. Such graphs also tend to be multiple orders of magnitude larger than the ones presented by Nazi et al. Also, we would like to generalize over graphs with different underlying degree distributions,

which might also be graphs from the same application domain (but with different degree distributions not as in Nazi et al.), but also graphs of totally different origin.

3.3.2 Deep Learning and Spectral Embedding for Graph Partitioning

Another approach to applying deep learning methods to conventional graph partitioning is described by Gatti et al. [11]. It represents a modification to the previously explained GAP model [10] and is the first of its kind to bisect non-attributed graphs. It takes a more global approach compared to GAP by using spectral graph partitioning theory. Still, it also subdivides the model into an embedding and a partitioning module. Both modules incorporate a multilevel algorithm inspired by the multigrid algorithm [43].

The goal of the embedding module is to predict an approximate Fiedler vector f [44], the eigenvector to the smallest non-trivial eigenvalue of the normalized graph Laplacian matrix $\widetilde{L} = D^{-1}(D - A)$ with D being the degree matrix and A the adjacency matrix. From spectral graph partitioning theory, we know that $\widetilde{L}$ can be used to express the normalized cut objective as

$$f^T \widetilde{L} f = \frac{4|\{\{u, v\} \in E \,|\, u \in V_1, v \in V_2\}|}{\sum_{i \in V_1} \deg(v_i)}$$

and that the Fiedler vector in the continuous relaxation of the problem minimizes this term. Therefore the output of the embedding module should already give an approximate Fiedler vector and therefore an approximate spectral graph partitioning.

First, the input graph is coarsened using a heavy edge matching algorithm until only two nodes are left. On every level, the matrix $F \in \mathbb{R}^{n \times d}$ is predicted holding the eigenvectors to the d smallest eigenvalues (including 0) for the graph having n nodes on the current level. Gatti et al. sets $d = 2$ such that f_1 is the constant one vector $\mathbb{1}$ and f_2 represents the Fiedler vector. The prediction on each level is done using multiple layers of PyTorch Geometric's SAGEConv layer [45], an implementation of GraphSAGE's sample and aggregate methodology. Nevertheless, the aggregator used is different from the ones introduced in Section 2.6

$$h_v^k \leftarrow Tanh((W_1 h_v^{k-1} + W_2 \cdot Mean(\{h_u^{k-1}, \forall u \in N(v)\})).$$

with h_v having just two components. Input to the SAGEConv layers is the interpolated matrix F from the previous layer, which is chosen as the identity matrix on the very coarsest level. The exact same SAGEConv layers are used on every level of the multilevel algorithm, such that the trainable parameters remain independent of the graph size. On the finest level, several dense layers are applied before standardizing the final embeddings F using QR-factorization. The loss function used for training the embedding module minimizes the residual to the eigenvectors of the normalized graph Laplacian matrix and minimizes the total cut size by minimizing the eigenvalues of the normalized graph Laplacian matrix:

$$\mathcal{L} = \|\widetilde{L}F - \Lambda F\| + \sum_{i=1}^{d} \lambda_i.$$

The eigenvalues are computed as $\lambda_i = F_{:,i}^T \widetilde{L} F_{:,i}$ with F being orthogonal and normalized after the QR-factorization and $\Lambda = diag(\lambda_1, \ldots, \lambda_d)$ being the diagonal matrix of the eigenvalues. Before passing the Fiedler vector as the approximate spectral partitioning to the partitioning module, it is again normalized according to

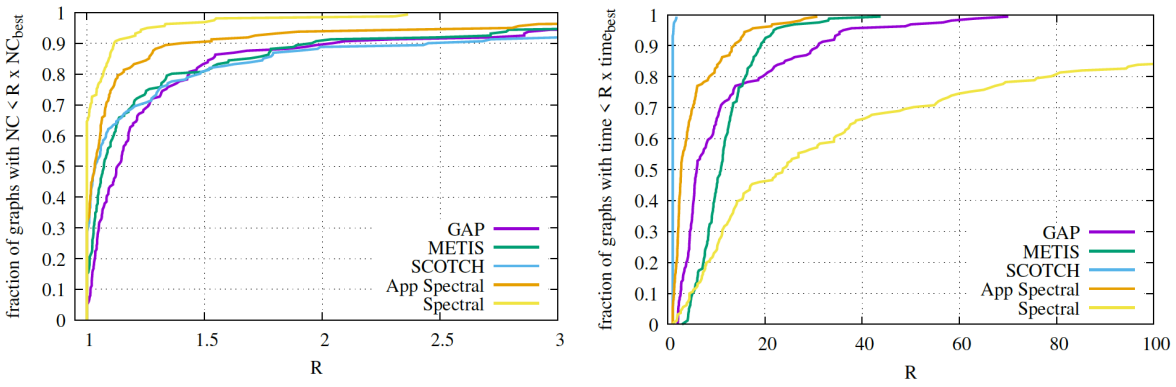
$$F_{:,2} = f_2 = \sqrt{|V|}(f_2 - \frac{\sum_i^{|V|}(f_2)_i}{|V|})$$

to have zero mean and unit variance.

The partitioning module uses the provided matrix of eigenvectors F and assigns each node the corresponding row of F. Again a multilevel scheme is applied, this time also applying several layers of the aggregator function before every coarsening step using the heavy edge matching heuristic. On every level, the feature tensor F^l is first stored and then passed from the coarsening routine to the uncoarsening routine on the same level. The interpolated feature tensor $\widetilde{F}^l$ coming from the coarser level subtracts the residual error from F^l such that the approximate Fiedler vector $F_{:,2}^l$ is refined more and more to approach the optimum. Again the same SAGEConv layers are applied on every level to keep the model size fixed. On the finest level again several dense layers are applied before creating block assignment probabilities for each node using the softmax function. The loss function used for training the partitioning module is the same as from Nazi et al. [10] but neglects the balance term.

Summarizing the architecture, the embedding module alone learns to approximate a spectral graph partitioning and the partitioning module tries to correct the errors of the embedding module regarding the spectral graph partitioning and outputs a final probability distribution for every node. Looking at the output of the embedding module alone can give an approximate spectral bisection by assigning $V_1 = \{v_i \in V \mid F_{i,2} < c_i\}$ and $V_2 = V \backslash V_1$ with c_i chosen as the $F_{i,2}$ minimizing the normalized cut or the median of $F_{:,2}$ for balanced partitions.

The experiments report a decent approximation of the exact spectral partitioning from the embedding module with the worst performing graph instance having a 7% higher normalized cut and 11% higher imbalance. The overall model is indeed capable to outperform METIS [46] and Scotch [47] on some chosen graph instances. It also corrects the high imbalance that would have been produced by an approximate spectral partitioning from the embedding module on another graph instance. Nevertheless, not always does the final model give the desired result.



(a) Normalized Cut

(b) Time

Figure 3.2: Performance profile ratios for the SuiteSparse instances [11, 48]

In Figure 3.2 by Gatti et al. the purple line indicated as "GAP" is the final model from Gatti et al. We see that the final model cannot always correct the errors from the embedding module. The curve from the approximated spectral partitioning that the embedding module outputs is consistently over the one from "GAP". Regarding the running times, the final model as expected always runs faster than true spectral partitioning and in most cases also faster than METIS, but slower than Scotch. Gatti et al. also state well generalization capabilities of the final model without much further investigation.

3.4 Streaming Graph Partitioning using Deep Learning

Inspired by GAP from Nazi et al. [10], and based on the preliminary work by Zwolak [49], Abbas [12] and Zwolak et al. [13] introduce the first and to the best of their and our knowledge only online graph partitioning deep learning model called GCNSplit. Traditional streaming graph partitioning algorithms like High Degree Replicated First (HDRF) from Petroni et al. [50] make use of an unbounded state of size $O(|V|)$, which is why they are impractical for unbounded streams as the allocated memory is growing and needs to be updated frequently. GCNSplit on the other hand is a stateless fixed-size model and is able to handle unbounded graph streams.

In Abbas [12], GCNSplit is applied to unbounded edge streams. Its architecture is basically equal to GAP and based on an embedding module and a partitioning module. As the embedding module GraphSAGE with the max pooling aggregator and just two aggregation layers is used to encode attributed nodes into the latent embedding space. The partitioning module is a 3-layer fully-connected network with a final softmax layer to output block assignment probabilities. The dimensionality of the hidden layers for both modules is set to 64. The input feature vector of each node can have up to 2 500 components in the experiments performed by Abbas (see Twitch graphs). When an edge is streamed, both edge endpoints' feature vectors are fed consecutively (unsupervised model) or concatenated (supervised model) into GCNSplit to give a prediction. The probabilities received are input to the final block assignment heuristic[1]. Here Abbas differentiates between the *HighestOrLeastLoaded* heuristic, which assigns an edge to the block receiving the highest probability or to the least loaded block in case the balance constraint would get violated, and the *HighestAvailable* heuristic, which always chooses the most probable block which is not overloaded yet. In the case of the unsupervised model, where both nodes are fed consecutively into the model, the block is determined by the highest of both nodes' output block probabilities. Interestingly, Abbas reports a 50 - 226% higher imbalance of GAP compared to GCNSplit which extends GAP by its balance heuristic. Such high imbalances are not directly reported by Nazi et al. [10].

GCNSplit is trained offline in either a supervised or an unsupervised manner. Independently from the training mode, both modules are trained jointly using the same

[1] In the following always referred to as balance heuristic

loss function and mini-batch gradient descent. The unsupervised training is performed using the same loss function as GAP [10], but additional coefficients α for the expected normalized cut term and β for the balance error term resulting in

$$\mathcal{L} = \alpha \sum_{reduce-sum} (Y \oslash \Gamma)(1-Y)^T \odot A + \beta \sum_{reduce-sum} (\mathbf{1}^T Y - \frac{n}{k})^2.$$

For supervised training labels from the HDRF edge streaming graph partitioning algorithm [50] are used. The standard cross-entropy loss function is used to optimize the model. The training graph is typically a fixed-sized snapshot of the evolving target graph that should be partitioned online during prediction, but it can also be a different graph that has the same node features and a similar structure as the target graph. GCNSplit's generalization capability can therefore be compared to GAP [10]. Regarding the partitioning quality and applicability, Abbas notes:

> "As a result, we expect GCNSplit to be particularly effective on graph streams whose structural characteristics and feature distribution remain relatively stable over time. Nevertheless, if GCNSplit is applied on a graph stream with major concept drift, it will - in the worst case - behave like hash partitioning."

Hash partitioning is a stateless streaming graph partitioning algorithm that assigns edges with unique identifiers uniformly at random to blocks by applying a hashing function onto the edges. This typically leads to high replication factors.

In the case of the unsupervised model, online edge partitioning inference is performed by predicting the block assignments of both edge endpoints consecutively. On the other hand, the feature vectors of the nodes are concatenated and fed together into GCNSplit in the case of the supervised model, which was trained to output edge assignment probabilities instead of node assignment probabilities because of using HDRF's edge partitioning labels for training.

The experiments regarding partitioning quality show, that GCNSplit can well keep up with the HDRF baseline algorithm while being a fixed-size stateless model[2]. The unsupervised model turns out to work better than the supervised model as producing lower replication factor scores for the edge partitioning setting. Also, the HighestOrLeastLoaded balance

[2]While GCNSplit takes just 115KB to partition a large random graph with 1.3B edges, 930M nodes, and 64 random features per node, HDRF would need more than 116GB.

heuristic consistently achieves better results than the HighestAvailable balance heuristic. Applying the trained model to a different graph of the same application domain, i.e., training on the Twitch-DE graph and applying the model to the Twitch-ES graph during prediction for instance, still results in high-quality partitions. Nevertheless, once the evolving graph develops further away from the snapshot the model was trained on, partitioning quality also decreases. GCNSplit only works in environments with stable graph structures and feature distributions over time. Else, Abbas also states that online training would need to be necessary to counter this issue.

To summarize, GCNSplit is an extension of GAP [10] to be applicable for unbounded edge streams. It introduces new balance heuristics to enforce hard balance constraints and also generalizes to unseen graphs, but is limited to graphs with a similar structure and originating from the same application domain.

What remains is to engineer a model and streaming approach, that is capable to partition non-attributed, featureless graphs. This way, one could also enlarge the area of generalization capability beyond just graphs from the same application domain. What needs to be evaluated is, whether graphs of different structures and origins can also be partitioned using such a machine learning or deep learning model as a heuristic.

CHAPTER 4

Streaming Graph Partitioning Framework

4.1 Overview

For the task of streaming graph partitioning, we first introduce our buffered streaming model in Section 4.2. It supports modes for both node and edge streaming, whereas our experiments in Chapter 5 mainly focus on node streams. Once the nodes or edges are streamed into the buffer, we need to build features for each node, which are necessary to make predictions about the block assignment of each node or edge. Therefore, we introduce different groups of features in Section 4.3 that we will later on empirically evaluate for each model. As a core conceptual novelty that comes with our framework, prediction propagation is explained in Section 4.4. It serves to propagate information through the buffer to build better features that consecutively result in better partitions. The different machine learning models, their configuration and architecture are introduced in Section 4.5. Last but not least, two balance heuristics inspired by Abbas [12] and Zwolak et al. [13] are presented to ensure balanced partitions in Section 4.6.

The high-level structure of the streaming graph partitioning routine for node streams, that we mainly focus on in this work, is given in Algorithm 3. After having built the buffer, we stream nodes in a sliding-window based fashion by the natural order of nodes. Every time new nodes are added to the buffer, they are added to the prediction propagation queue.

For all nodes inside this queue, feature vectors are (re)generated, optionally normalized and standardized, and predictions regarding a block assignment are done while taking balance heuristics into consideration. In the case that the block assignment changes or is initially set, neighboring buffer nodes are added to the prediction propagation queue of the following round until all specified prediction propagation rounds are performed. This continues until the entire graph is streamed and partitioned.

Algorithm 3 Structure of the streaming graph partitioning routine

```
Build initial buffer
while Graph not entirely streamed do
    Add step size new nodes to the buffer and the pred. prop. queue
    Optional: Enrich buffer nodes with additional features
    for Pred. prop. rounds do
        while Current pred. prop. queue not empty do
            Pop node from current pred. prop. queue
            Construct the node's feature vector
            Optional: Normalize and standardize the feature vector
            if Feature vector changed then
                Assign new feature vector to node
                Predict the node's block while considering the balance heuristic
                if Block assignment changed then
                    Add neighboring buffer nodes to the next pred. prop. queue
    Update the buffer
```

For a more visual explanation of the prediction phase of node streams, Figure 4.1 can be used. It represents a streaming graph partitioning scenario with $k = 4$ perfectly balanced blocks. The buffer contains 16 nodes and the step size is chosen as four nodes at a time. As the previous two blocks, disregarding the way how they were built, already reached their maximum load, node 25 is given a new block assignment by the chosen balance heuristic. Subsequently, node 21 is added to the next prediction propagation queue again, as it might want to choose a better block assignment. The next node in the current prediction propagation queue is node 26. Auxiliary nodes in the current buffer are never added to a prediction propagation queue.

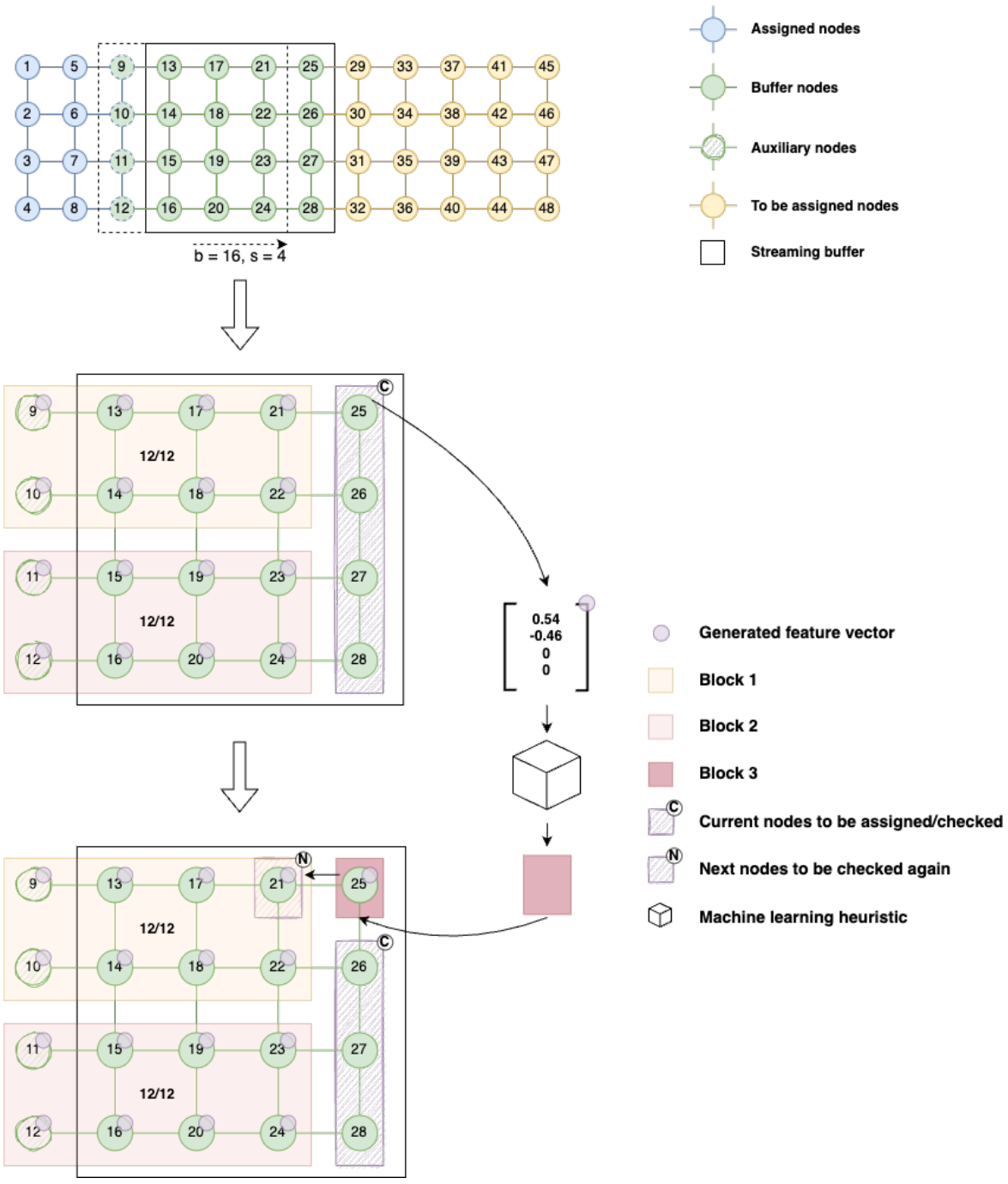


Figure 4.1: Overall streaming algorithm during the prediction phase. We use a sliding-window based streaming algorithm with a fixed buffer size $b = 16$ and a fixed step size $s = 4$. Once new nodes are added to the buffer, their features get constructed and block assignment predictions are made while regarding balance heuristics to ensure a hard balance constraint. Subsequently, the newly predicted block assignments are propagated to neighboring buffer nodes, which might improve their feature vectors and the block assignment predictions built upon these feature vectors.

4.2 Buffered Streaming Model

The node stream is implemented in a sliding window based buffered streaming fashion. The scalar s stands for the *step size* in this model, whereas b denotes the *buffer size*. Both scalars are parameters of the algorithm. Note that the problem allows us to have $O(|V|)$ space available.

We first introduce the buffer as being the subgraph based on b streamed nodes from the original graph. Every node has a constant number of attributes which are:

Feature	**Description**
nodeID	The ID of the node
weight	The weight of the node
feature_vector	The computed feature vector of the node used for the prediction
block	The predicted block of the node
true_block	The ground truth block of the node
auxiliary	Whether the node is an auxiliary node
community	The cluster ID resulting from a label propagation run on the buffer

Table 4.1: Attributes assigned to each node

All nodes are streamed in the natural order of their node IDs. We denote all b nodes currently inside the buffer as *current nodes* and all nodes removed from the buffer as *past nodes*.

Initially, b nodes are streamed into the buffer. The subsequently induced subgraph by these current nodes gives us a local view of the entire graph. Next, the training or prediction routine is run on the entire buffer, which partitions the current nodes in the buffer. For all subsequent steps, the s oldest nodes by node ID are removed from the buffer and the next s nodes are streamed into the buffer. Note that current nodes might have edges towards past nodes, which were already assigned to a block. We solve this by adding such past nodes as *auxiliary nodes* to the buffer again, without adding the edges between such past nodes to the buffer, but only those towards current nodes. This is especially important for the feature generation of each current node as some features are based on attributes of neighboring nodes.

After the current buffer is built, nodes can be enriched with additional information. Currently, the only information added are cluster IDs that result from a label propagation [51] algorithm run on the current buffer to build a clustering. The number of label propagation rounds is a parameter of the algorithm.

The buffered streaming model for edge streams is equivalent to the one for node streams. The buffer contains b edges at a time and removes/adds s new edges on every streaming step ordered by ascending edge IDs. Auxiliary edges are added between nodes inside the buffer and nodes outside the buffer, that were already assigned a block. The only difference is, that the feature_vector attribute is still attached to every node as the block assignment prediction for every edge is performed by investigating both edge endpoints to conclude the final block assignment prediction of an edge.

During streaming, we also maintain a *partition state*, which stores the number of nodes or edges in each block (*block count*) and the blocks recent individual nodes or edges were assigned to (*partitioning history*). The size of the partitioning history buffer can be chosen arbitrarily as a parameter of the algorithm.

4.3 Feature Engineering

As we will mainly focus on node streams in the remainder of this work, we will exclusively explain the following features in the context of node streams but all of them can be easily mapped to edge streams as well. Once we use edge streams in Section 5.5.4, we will disclose how to adapt the chosen features.

To predict the block assignment of a node, each node is represented by a feature vector that needs to be calculated beforehand. We define three different kinds of feature groups, i.e., *statistical features*, *greedy features* and *heuristic features*.

Statistical features. Statistical features are divided into commonly known node characteristics (abbreviated NC) and the partitioning history (abbreviated PH). Common node characteristics used are degree, local clustering coefficient, node clique number, node ID (which is also the index in the stream when streaming in natural order), percentage of nodes streamed (to give the node ID a reference value), and the cluster ID that results from running the label propagation algorithm in the buffer before calculating the feature

vectors. The local clustering coefficient describes how much a node's direct neighborhood is connected and is calculated as $C_v = \frac{2L_v}{deg(v)(deg(v)-1)}$ with L_v being the number of links between directly neighboring nodes. The node clique number tells the size of the largest maximal clique containing an observed node. To add the partitioning history, we append a vector of size k counting the number of latest assignments to each block from the partitioning history. The statistical features result in a vector of size $k + 6$.

Greedy features. The first feature (abbreviated NBC) regards the community a node gets assigned to after label propagation. By only looking at the direct neighbors of a node that were assigned to the same community, a k-vector counts for each block the number of neighbors in the same block. The second feature (abbreviated EC) lists for each block the number of added cut edges when a node gets assigned to this specific block. The last k-sized feature (abbreviated NB) simply counts for each block the number of neighbors that were assigned to this block.

Note that the greedy features are based on previous block assignment predictions. Therefore, in case the prediction does not perform well, the subsequently calculated features building upon such predictions will amplify bad prediction performance. Also, the greedy features become more and more accurate the more the direct neighborhood of a node is explored, meaning the more neighbors have already been partitioned. To make features and subsequent predictions more accurate, the concept of prediction propagation is introduced in Section 4.4.

Heuristic features. Here we use the two heuristics Fennel and LDG, which are introduced in Section 2.2. Both of them will result in a k-vector with each component storing the score of the heuristic for the specific block. Just like the greedy features, the heuristic features also become more accurate through prediction propagation as they also depend on the block assignments in the direct neighborhood.

4.4 Prediction Propagation

As stated in Section 4.3 some features depend on the block assignments of a node's direct neighborhood. The less the block assignments of the direct neighborhood are already done, the more difficult it is to get accurate features based on this information. The concept of *prediction propagation* now has the goal to improve on previous block

assignment predictions for current nodes inside the buffer in case the feature vectors of such nodes become more accurate by having more partitioned neighbors.

Once a current node changes or initially receives its block assignment, all neighbors of this node in the buffer, except auxiliary nodes, are added to the queue for the next prediction propagation round. Then in the next round, in case the recalculated feature vector for such a node popped from the queue is now different, its block assignment is predicted once more. If the block assignment prediction changes, again the neighbors of such a node are added to the queue for the next prediction propagation round. This continues for several custom-defined rounds. Note that the more prediction propagation rounds are performed, the longer a single streaming step takes. All auxiliary nodes inside the buffer are excluded from prediction propagation, they just serve to make their block assignment decision available for the feature construction of their direct neighbors, that are true members of the buffer and whose prediction should possibly improve.

Also, this principle is based on the assumption that the nodes being streamed by their natural order are not of totally different origins inside the graph. There should be some locality included in the node stream which will then be reflected also into the buffer. If this is not the case, then predictions could not be propagated appropriately and therefore the partitioning quality would suffer.

In the context of edge streaming, prediction propagation means adding outgoing edges from the source and target node of an edge, whose new block assignment should be propagated through the buffer, to the prediction propagation queue of the following round.

4.5 Machine Learning Models

We chose to approach the problem using both models from conventional machine learning and more advanced deep learning. The conventional models have the advantage to be faster to train and more explainable. The deep learning models have more degrees of freedom and therefore need longer to be trained, but should perform more accurately. Also, they should be better capable of extracting patterns, feature interactions, and partitioning heuristics as they theoretically can approximate any given function according to the universal approximation theorem. We will introduce all models in the following.

Baseline. The baseline algorithm is an implementation of the Fennel heuristic (see Section 2.2) incorporated into our framework. The block with the maximum Fennel score gets assigned the currently to be predicted node or edge. This baseline algorithm is used to compare the machine learning heuristic against the Fennel heuristic.

Logistic Regression. The Logistic Regression model is the simplest classifier from conventional machine learning techniques. It performs especially well on linearly separable data. We use the implementation from the scikit-learn[1] library [52]. We configure the Logistic Regression classifier to use the SAGA solver, a variant of the stochastic average gradient solver, L2-regularization, and a convergence tolerance of 10^{-4}. The best value for the inverse of the regularization strength C is evaluated in Section 5.4.3.

Gradient Boosted Decision Trees (GBDTs). As a parameter-free model GBDTs make no assumptions about the given data. Through boosting they can fit very complex functions very closely which makes them one of the most popular conventional machine learning classifiers. We use the scikit-learn wrapper implementation from XGBoost[2] [20]. The GBDTs hyperparameters are mainly studied in Section 5.4.3, i.e., the number of estimators to use, the learning rate, the maximum depth of each tree estimator, the subsampling percentage, and the size of the regularization parameter. No early stopping is performed.

Support Vector Machine (SVM). The SVM, another representative of the conventional machine learning models, was the main competitor of most neural network models when their popularity increased again due to the increasing amount of data and computing power in the early 2000s. The model's implementation is also taken from the scikit-learn library [52]. The hyperparameters, namely the kernel and regularization parameter, are also studied in detail in Section 5.4.3. The convergence tolerance is set to 10^{-4}.

GraphSAGE. In Chapter 3, we mention that GraphSAGE [32] is often used to learn node embeddings that are used to make block assignment predictions using a separate partitioning module as a classifier. Nevertheless, GraphSAGE alone can also serve as a classifier by squashing the final embeddings through a softmax layer to output block assignment probabilities for each node. After experimentally validating the architecture

[1] https://scikit-learn.org/stable/
[2] https://xgboost.readthedocs.io/en/stable/python/

of GCNSplit from Abbas [12], we take over its configuration parameters. The model is built using PyTorch Geometric[3] [45]. We use two 64-unit SAGEConv layers with max-pooling aggregators, enabled normalization, and a ReLU non-linearity in between. The model outputs log-probabilities using the log-softmax function.

Partitioner. The Partitioner model adds an additional partitioning module on top of the previously introduced GraphSAGE model. It is a dense neural network with three 64-unit linear layers and ReLU non-linearity in between and a final log-softmax layer to output the block assignment probabilities. Again the architectural configuration was taken from Abbas [12] after experimental evaluation. The partitioning module was built with PyTorch[4] [53].

The GraphSAGE and Partitioner model are optimized over 2 500 epochs using the Adam optimizer [54] and a learning rate of 10^{-5}. No weight decay is enabled. As we output log-probabilities to stronger punish incorrect predictions and for more numerical stability, we also use the negative log-likelihood loss function for supervised training. We additionally experimented with the loss function from Nazi et al. [10] for unsupervised training, but the convergence time takes far too long on our machine. Even if Abbas [12] reports the unsupervised model to perform better than the supervised model, we believe that this is due to improvable block labels in the ground truth data as normally supervised learning tends to work better than unsupervised learning in a direct comparison. We generate block labels using the sophisticated offline multilevel graph partitioner KaFFPa [14] in the "eco" configuration. No special weight initialization techniques like Xavier initialization [55] or He initialization [56] are performed, as the models are converging fast already.

4.6 Balance Heuristics

Inspired by Abbas [12] and Zwolak et al. [13], we also define the same two balance heuristics applied to node and edge streaming graph partitioning to ensure the balance constraint when assigning nodes or edges to blocks. For edge partitioning, the received block probabilities for both edge endpoints are considered together. The first *MostProbable* heuristic always assigns a node or edge to the most probable block given by the machine

[3] https://pytorch-geometric.readthedocs.io/en/latest/
[4] https://pytorch.org/docs/stable/

learning heuristic that does not violate the balance constraint. The second *LeastLoaded* heuristic tries to assign a node or edge to the most probable block and in case of an overload to the least loaded block. Both heuristics apply random tie-breaking in case two blocks have an equal probability to get the current node or edge assigned.

CHAPTER 5

Experimental Evaluation

5.1 Overview

After having introduced our framework, we now come to the experimental evaluation. In Section 5.2 we first introduce the two datasets used and the machine on which the experiments are run. Also, the initial streaming configuration is presented. Note that up to Section 5.5.4, we exclusively focus on node streams. The way the training routine works and the different evaluation methodologies are the subject of Section 5.3.

At the beginning of the tuning phase of Section 5.4, we figure out which features work best for which model and deep dive into the features of every feature group in Section 5.4.1. Next, we analyze whether feature vectors should be preprocessed using feature normalization and standardization in Section 5.4.2. Each model's hyperparameters will be tuned in Section 5.4.3. Besides the model tuning, also the two balance heuristics are compared to each other regarding their performance in Section 5.4.4. When tuning the streaming mode in Section 5.4.5, we especially focus on the buffer size b, the step size s, and the number of prediction propagation rounds. We also do an excursus into a BFS based streaming order and its impact on the partitioning quality. Throughout the tuning phase, decisions about the configuration parameters of the framework are made as a conclusion of every section. These configuration parameters then remain fixed for all upcoming experiments.

Once the tuning is done, we continue with the final evaluation phase in Section 5.5 and first evaluate the generalization capabilities of the different models in the framework in Section 5.5.1. Here we differentiate between two types of generalization. Instance generalization focuses on generalizing to unseen nodes in evolving graph scenarios and group generalization serves to generalize to even unseen graphs. In both scenarios, the models are compared against the Fennel based baseline algorithm. Next, the models' behaviors for different imbalance settings are studied in Section 5.5.2, followed by running time experiments regarding training and prediction time in Section 5.5.3. Last but not least, a comparison to the main machine learning based and an algorithmic competitor for edge partitioning will be drawn in Section 5.5.4.

5.2 Experimental Setup

To run the experiments we select two disjoint datasets, a small one for tuning the models and the streaming mode configuration and a bigger one for the final evaluation and the comparison against the competitor GCNSplit. Each dataset contains graphs from different groups of graphs, e.g., social graphs, road networks, meshes, etc. Details about both datasets can be found in Appendix A. The different groups of graphs also represent different types of degree distributions, i.e., Gaussian-like, more regular, and power-law based degree distributions.

The graphs originate from the 10th Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) implementation challenge [57], the well-known Walshaw benchmark [58], the "network repository" [59], the Stanford Network Analysis Project (SNAP) [60] and the laboratory for web algorithmics [61].

The artificially generated graphs, i.e., random geometric graphs and graphs from Delaunay triangulations, are generated by randomly placing points in the unit square. All graphs were preprocessed such that the final graphs are undirected with unit node and edge weights. Also, parallel edges and self-loops were removed. Isolated nodes are supported.

For the training phase, we use a machine with two six-core Intel Xeon E5-2630 v2 processors running at a 2.8 GHz base clock and 3.1 GHz turbo clock. It has 64 GB of main memory and 3 MB of L2-Cache. The machine runs Ubuntu GNU/Linux 20.04.1 and has the Linux kernel version 5.4.0-48-generic.

For the initial experiments on the tuning dataset, we choose a buffer size of 256, a step size of 16, and three prediction propagation rounds[1]. The datasets used allow us to stream from the main memory, future work should include experiments with huge graphs that are streamed from the disk. After every streaming step, five label propagation rounds are performed to build the clusters needed for some features. Initially, every node is its own cluster, and the cluster IDs from previous streaming steps are always reused for the current step. As a balance heuristic, we assign nodes to the most probable, still feasible block. The ten last block assignments are stored in the partitioning history. Note that the streaming configuration mentioned is also investigated and refined further in Section 5.4.5.

We perform the tuning phase for a difficult problem setting with $k = 16$ blocks and $\epsilon = 0.0$ imbalance. As the remaining experiments in the evaluation phase are more relaxed, we hope to prepare the algorithms well enough for any such problem setting. The evaluation phase allows a bigger imbalance by adding experiments with $\epsilon = 0.03$ imbalance and also investigates simpler partitioning procedures into fewer blocks.

5.3 Methodology

Each model is trained after performing a streaming routine. During training, the streaming routine only serves to build the features, the predictions are replaced with the ground truth block labels computed by KaFFPa [14]. The buffered training data is then used all at once to train the model after the streaming routine. This approach is again only possible if the training data fits into the main memory, else one needs to consider incremental learning or out-of-core learning techniques, which are not supported by all models.

To average the total cut size results we use the geometric mean. This minimizes the effect of extreme outliers on the average total cut sizes as not all graphs are located on the same scale regarding the number of edges.

Throughout the tuning phase, we often utilize a concept called t-Distributed Stochastic Neighbor Embedding (t-SNE) [62]. This technique is used to visualize high-dimensional data in just two or three dimensions. Similar points in the original space will also be

[1]The small road-euroroad graph only allows a buffer size of 128 and a step size of 8.

similar and nearby in the visualization after applying a t-SNE dimensionality reduction. We use this concept to visualize the feature space in order to evaluate if features give well-separable clusters of different classes.

In the evaluation phase, we often utilize performance profile plots among others. Such a plot compares the performance of the desired algorithm against a baseline algorithm regarding the total cut size. It shows how many instances in percent are within a factor of R away from the total cut size reported from the baseline algorithm on a per-instance level. The more the curve is pushed to the upper left corner, the better the evaluated algorithm. Note that the plot is especially thought to be used in the case it is already expected that the algorithm performs far worse than the baseline algorithm, e.g. if the baseline algorithm is an exact algorithm or the best currently known algorithm.

5.4 Tuning Phase

5.4.1 Feature Selection

Feature selection plays a major role in many supervised machine learning tasks. In the following series of experiments, we evaluate which set of features works best on which models while comparing to the Fennel based baseline algorithm.

In the case of conventional machine learning models, the prediction success heavily depends on which features are given to the model. These features need to be directly indicative and characteristic towards which block to assign a node. If not, the model will not be able to learn appropriate decision rules that separate the blocks from each other. We therefore evaluate the hypothesis, of whether the greedy and heuristic features perform notably better than the statistical features.

Contrary to conventional machine learning models, deep learning models have the claim to learn features on their own by just giving the raw data as the input. These models should be capable to learn partitioning patterns and heuristics by themselves without guidance, just as Convolutional Neural Networks (CNNs) learn structural patterns in images to make a classification prediction. Therefore, statistical features should become more important and helpful for the GraphSAGE and Partitioner model, even though

not directly indicative towards the blocks. This is to be investigated empirically in this section.

We run the experiments for the feature selection on the tuning dataset with $k = 16$ blocks and $\epsilon = 0.0$ imbalance. We train on 80% of the nodes uniformly sampled between the blocks and test on the remaining 20%. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes. The (+) symbolizes an increase in total cut size, (−) a decrease in total cut size, and (=) an equal total cut size, always comparing against using all features of the group. Each experiment on each graph is repeated five times using different random seeds.

The feature selection experiments have the same structure over all models. First, we compare the performance of the different feature groups against each other when using all of their contained features. We then specify which features are the most significant in each feature group. Lastly, we compare the selected features from each group against each other and agree on the feature(s) to be used for each model. All reported total cut sizes can be found in Appendix B.

Logistic Regression. We initially set the inverse of the regularization strength $C = 1$ for the feature selection experiment, which is the default value chosen by scikit-learn. Looking at the initial feature group comparison, we can already see that the hypothesis made for the conventional machine learning models holds true for the Logistic Regression classifier (see Table B.1). On average the statistical feature group performs far worse than its greedy and heuristic feature group competitor. Nevertheless, on the email-EuAll graph and partly on the delaunay_n14 and cond-mat graph the statistical feature group performs better.

In the statistical feature group, we can conclude that the feature based on the partitioning history (PH) is less helpful than the node characteristics (NC) feature vector (see Table B.2). Combining both features together nevertheless seems to give the best results. For the greedy feature group, we make the observation that only the edge cut (EC) related feature seems to give the best results (see Table B.3). Combining it with the second-best neighboring blocks (NB) feature also does not improve the performance. In the heuristic feature group, we recognize, that a feature based on the Fennel scores alone is far more helpful than a feature based on LDG scores (see Table B.4). We therefore also cut the latter out of the heuristic feature group.

If we now compare the selected best feature from each feature group, we obtain Table B.5. This makes us conclude that, for the Logistic Regression classifier, the Fennel score based feature gives the best performance, even slightly better than combining it with the greedy feature. Unexpectedly, already a simple Logistic Regression model using just the Fennel score based feature can outperform the Fennel based baseline algorithm on average. Eight out of ten graphs from the tuning dataset are best partitioned using this Logistic Regression model. So far the selected greedy and heuristic features perform notably better than the statistical features with the exception of email-EuAll, where the node characteristics (NC) feature outperforms the selected features of the greedy and heuristic feature group in a direct comparison. Also on delaunay_n14 the statistical features outperform the Fennel score based feature.

GBDTs. We repeat the same process now for the GBDTs. We initially use 100 estimators with a maximum depth of three and a learning rate of 0.1. Every estimator is trained on a 50% subsample of the original training dataset, the regularization parameter λ is set to one. Again we see that the greedy and heuristic feature perform better on average than the statistical features, with the exception being again delaunay_n14 and email-EuAll, but also astro-ph, where the greedy and heuristic feature group perform worse (see Table B.6). Still, the average analysis supports our hypothesis made at the beginning of this section.

Again the statistical feature based on the partitioning history (PH) does only convince to be used in practice when combined with the node characteristics (NC) feature as this gives the best reported total cut size for this feature group (see Table B.7). We recognize slightly different behaviors in the greedy feature group, where non of the features can outperform a combination of the edge cut (EC) and neighboring blocks (NB) related feature (see Table B.8). The heuristic feature group evaluation again shows the Fennel score based feature to perform best (see Table B.9).

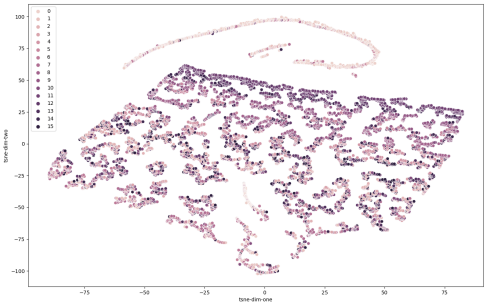
This also holds true when comparing the Fennel score based feature to the other winners of each feature group (see Table B.10). Even combining the best features from the greedy and heuristic group does not give a performance improvement to just using the Fennel score based feature alone. So also for the GBDTs classifier, we have to conclude that the statistical feature group performs notably worse than the other two groups. An exception is again the email-EuAll graph, where the node characteristics (NC) feature outperforms the selected features of the greedy and heuristic feature group. Also do the statistical

features outperform the Fennel score based feature on the delaunay_n14 graph and the greedy features on the astro-ph graph. Again seven out of ten graphs are partitioned best using the GBDTs instead of the Fennel based baseline algorithm.

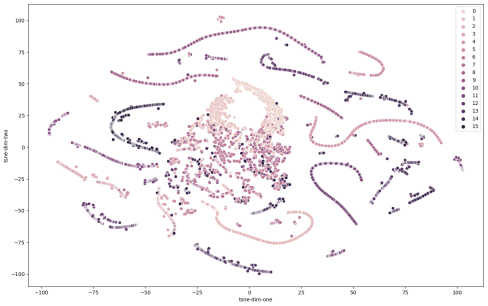
SVM. For the SVM, we choose a generic configuration with a polynomial kernel of third degree with the regularization parameter set to $C = 1$ (and the kernel coefficient γ set to the default configuration, i.e., $1/(\#features \cdot Var(X))$). Also, we cap the number of training nodes to 10 000 as the quadratically growing training effort through the kernel matrix would otherwise explode. Surprisingly, the statistical feature group starts to perform far better compared to being used with the Logistic Regression or GBDTs classifier (see Table B.11). It even outperforms the greedy feature group on average, which is why the initial hypothesis cannot be proven for the SVM. Nevertheless, the heuristic feature group still results in the lowest geometric mean over the total cut size.

As expected from the previous experiments, using node characteristics (NC) seems to work better than using the partitioning history (PH) as a feature (see Table B.12). Nevertheless, a combination of both gives better total cut size results than using each of the features on their own. Looking at the direct comparison between the single features of the greedy feature group, we conclude the pure edge cut (EC) related feature to be the best choice this time (see Table B.13). An ablation study on the heuristic features again results in the tendency to use pure Fennel scores as features instead of using Fennel or/and LDG scores combined (see Table B.14).

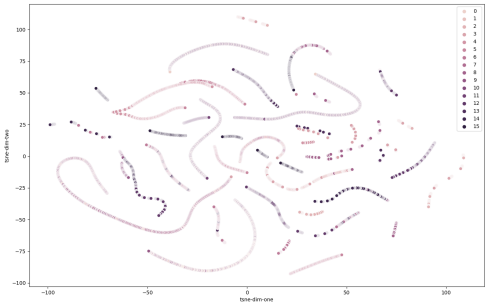
Comparing the best features of all feature groups against each other results again in the Fennel score based feature being recommended to be used, nevertheless also the features from the statistical group win on some graphs in a direct comparison against the Fennel score based feature, namely on the graphs from the artificial and social group (see Table B.15). Looking at the results on the email-EuAll graph for instance, the statistical features still perform far better.



(a) NC+PH on email-EuAll



(b) Fennel on email-EuAll



(c) Fennel on fe_sphere

Figure 5.1: SVM feature evaluation: T-SNE visualizations of different feature spaces

By looking at the t-SNE visualizations of the feature spaces in Figure 5.1, we also cannot directly tell why the statistical features perform better on the email-EuAll graph, no homogeneous clusters can be determined. But visualizing a high dimensional feature space with just two dimensions might simply be too difficult, such that the clusters from the higher dimensional space vanish[2]. Looking at the feature space build from the Fennel scores, we can already better recognize cluster-like structures and lines for the email-EuAll graph even though the Fennel score based feature performs worse regarding total cut size. Clear cluster-like structures on the other hand can be determined for the fe_sphere graph, where the Fennel score based feature performs far better than others in Table B.15.

[2]Note that the feature space itself might not have well separable point clouds by default, as these often first appear after applying the kernel trick.

When combining the statistical features with the Fennel score based feature, we do not obtain a performance boost. Nevertheless, using the Fennel score based feature, we are able to outperform six graphs using the SVM compared to the Fennel based baseline algorithm.

To summarize, our hypothesis that the greedy and heuristic features perform notably better than the statistical features turned out to be true for the majority of cases for the conventional machine learning models. Only for the SVM model the statistical features have a bigger impact and the scale-free email-EuAll graph also seems to be more accessible for streaming partitioning with statistical features than other graphs. But as adding the statistical features can also worsen the performance for other graphs, we agree on using solely the Fennel score based feature for the conventional machine learning models as it provides the best performance for the majority of graphs. We also need to note that so far we cannot strictly beat the Fennel based baseline algorithm without further tuning, nevertheless, such simple conventional models are already sufficient to beat the baseline algorithm on average.

GraphSAGE. Once more, we report the initial performance of the different groups of features on the tuning dataset, when using all features contained in each group (see Table B.16). So far the heuristic features have on average again the most promising outlook, nevertheless the importance of the statistical features has grown, the artificial graphs were best partitioned by using the statistical features for instance, and also on other graphs, models would prefer the statistical features over others. This would support our hypothesis that for deep learning models the statistical features can be used to extract patterns and heuristics for streaming graph partitioning.

Looking at the statistical features, different from the results for the conventional machine learning models, the partitioning history is more useful as a feature than plain node characteristics (see Table B.17). Nevertheless, combining both features together still tends to give the best results. A closer investigation of the greedy features shows favor towards the combination of the edge cut (EC) and neighboring blocks (NB) related feature (see Table B.18). Also for the last group of features, we again obtain similar results as for the previous models, there is a clear recommendation to use the Fennel score based feature over the LDG score based feature or a combination of both (see Table B.19).

Using GraphSAGE we are able to outperform the Fennel based baseline algorithm already in six out of ten graphs, not more than with the conventional machine learning models (see Table B.20). Still, the greedy and heuristic features outperform the statistical features on average, nevertheless the statistical features become more informative than in previous conventional machine learning models, especially for the artificial graphs, astro-ph, and email-EuAll. The deep learning model begins to find partitioning patterns and heuristics on its own from the raw input feature space, but which is still not enough to outperform the greedy and heuristic features. This would confirm our hypothesis that indeed statistical features become more important for GraphSAGE as the first deep learning model evaluated empirically in this series of experiments in contrast to the conventional machine learning models. Using a pure Fennel score based feature also performs better than combining it with the selected greedy features.

Contrary to the SVM a t-SNE visualization also does not need to have clear cluster-like structures anymore. The aggregator functions that GraphSAGE learns separate the hyperspace by far more than k hyperplanes. According to the universal approximation theorem, a neural network can divide a hyperspace into any arbitrary number of subspaces. When given enough degrees of freedom, it is in theory able to approximate any given function. This fact would also support the hypothesis of GraphSAGE being a model that is capable to handle also non directly indicative features like node characteristics (NC).

Partitioner. Again the experiment this time for the Partitioner model leads to the observation to use the greedy and heuristic features over the statistical features, even if again the impact of statistical features increases (see Table B.21). Just as for the GraphSAGE model, the artificial graphs are best partitioned using features from the statistical group, which again supports our hypothesis made for the deep learning models.

Looking at the statistical feature group first, we obtain that again the partitioning history (PH) feature wins over the node characteristics (NC) feature, but a combination of both still performs better than having each feature separated (see Table B.17). This is the same discovery as made for the GraphSAGE model. The conventional models prefer the node characteristics (NC) over the partitioning history (PH) in a direct comparison. For the Partitioner model, the neighboring blocks (NB) feature shows a tendency to be used over the other two greedy features or a combination of the edge cut (EC) related feature and the neighboring blocks (NB) feature (see Table B.23). Also for the last model in this

series of experiments, the Fennel score based feature prevails over the LDG score based feature or a combination of both features (see Table B.24).

Just as with the GraphSAGE model, the Partitioner model is also able to outperform the Fennel based baseline algorithm on average when using solely the Fennel score based feature. Only three out of ten graphs are still best partitioned using the baseline algorithm (see Table B.25). Supporting our hypothesis, the statistical features develop more importance for the Partitioner model, but which is still not sufficient to outperform the greedy and heuristic features on average. Nevertheless, on some graphs, the statistical features turn out to work really well, like on the artificial Delaunay graphs. Combining the Fennel score based feature with the neighboring blocks (NB) feature does not outperform the pure Fennel score based feature.

To summarize our hypothesis, indeed the deep learning models seem to be more suitable to extract patterns and heuristics out of the statistical feature group. The geometric mean of the total cut sizes across the experimental graphs is notably smaller than for the conventional machine learning models. Nevertheless, as the majority of graphs are partitioned best using the Fennel score based features, we decide to use this feature for future experiments with the deep learning models. This aligns with the decision made for the conventional machine learning models, that also use only the Fennel score based feature in the future. Just like the conventional machine learning models, the deep learning models are also capable to outperform the Fennel based baseline algorithm on average. So far nevertheless, the reported geometric mean of the total cut sizes based on the Fennel score based feature is lowest for the conventional machine learning models and not for the deep learning models. We will need to evaluate if this persists also after the tuning phase.

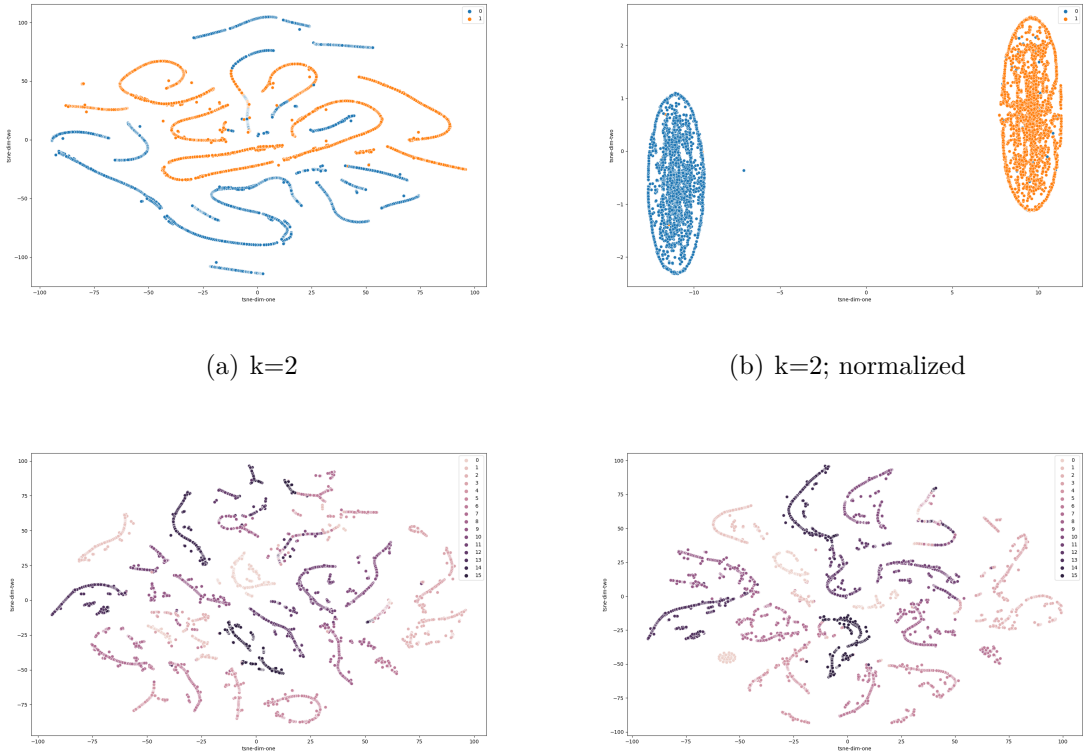
Side notes. We also want to note that the features from the greedy group are somewhat correlated, which can be seen in the reported total cut sizes that differ just a bit between the features independent from which model was used. Often the decision of which random seed is used decides which of the features wins the experiment. Especially if the neighboring blocks (NB) feature alludes to a specific partition, its matching component in the edge cut (EC) feature will be low. Also, the feature regarding the neighboring blocks from the same community (NBC) is just a more restrictive feature than the normal neighboring blocks (NB) feature.

The fact that on some specifically chosen graphs the statistical features are also helpful for the conventional machine learning models, should be based on the high variance of the degrees found in such graphs. The email-EuAll graph, which is often partitioned better with the selected statistical features than a greedy or heuristic feature, is a power-law based scale-free graph and therefore covers a wide range of degrees. Nevertheless, not all graphs whose degree distributions are power-law based, are partitioned best with the statistical features.

5.4.2 Feature Normalization and Standardization

In machine learning the quality of the data that is input to a model is essential for the model's performance. Preprocessing the input data might help the model to better recognize structures and patterns in it. Normalizing the data fixes the magnitude of the feature space to a specific expansion, which helps to make the model invariant to scaled data. This is especially important for generalization purposes over different-sized graphs as the previously selected Fennel score based feature can change in magnitude as it depends on the number of nodes n and edges m. Standardization transforms the data to have a zero mean and a unit standard deviation such that changes in the feature values have an equal impact on the model. Especially parametric models, that have an incorporated assumption about the data, are often dependent on such preprocessing techniques. The Logistic Regression classifier is such a parametric model as it assumes the data to be linearly separable. Also, the SVM tries to make the data linearly separable by using the kernel trick.

To help these models to perform better, we normalize the Fennel scores to be in the interval $[-1, 1]$ and apply standardization afterwards. In Figure 5.2 we can see which impact this has on the feature space of $k = 2$ blocks and $k = 16$ blocks. The t-SNE visualization for $k = 2$ blocks equals the exact feature space, whereas the t-SNE visualization for $k = 16$ blocks applies dimensionality reduction and puts such data instances close together that are also nearby in the original feature space.



(a) k=2

(b) k=2; normalized

(c) k=16

(d) k=16; normalized

Figure 5.2: Effect of feature normalization and standardization on the feature space: T-SNE visualization of the feature space of delaunay_n13

The non-preprocessed data results in a feature space, that is almost impossible to separate linearly. Whereas the Logistic Regression classifier would probably fail to do so, the SVM could still perform better by using the kernel trick. When normalizing and standardizing the data, one can now clearly separate the data linearly. Nevertheless, we still see that in such a feature space not all data instances can be classified correctly by using a simple linear model. Some data instances are put into the wrong cluster.

Parameter-free models on the other hand, i.e., GBDTs, GraphSAGE, and Partitioner, have more degrees of freedom and can more closely adapt to the feature space and try to correct the errors that the Fennel heuristic would generate in the feature space. Therefore it would not be beneficial to narrow down the feature space and make optimization harder, which is the case when normalizing the data. Instead, the original feature space

should be passed to such models that has more space to be optimized without forcing the models to overfit in order to correct the errors of the Fennel heuristic. Still, regularization needs to be applied to avoid potential overfitting.

5.4.3 Hyperparameter Tuning

The tuning of the hyperparameters is based on the previous feature selection results, meaning each model uses solely the Fennel score based feature. As each graph has its individual properties, the optimal hyperparameters will most probably also differ from one graph to another. This hypothesis will be evaluated for the tuning dataset, again for $k = 16$ blocks and $\epsilon = 0.0$ imbalance. Each of the experiments is performed using five-fold cross-validation. We use the entire graphs to find the optimal hyperparameters.

Logistic Regression. The only parameter that needs to be tuned here is the regularization parameter C, the inverse of the regularization strength. Therefore we run an exhaustive grid search on the set of $\{10^i | -3 \leq i \leq 1\}$ with the following result:

Graph	**C**
delaunay_n13	1
delaunay_n14	10
astro-ph	0.1
cond-mat	0.1
fe_4elt2	0.1
fe_sphere	0.1
road-minnesota	0.01
road-euroroad	1
email-EuAll	0.1
wordassociation-2011	1

Table 5.1: Logistic Regression hyperparameter evaluation: Regularization C

We see that a clear pattern between the graphs' structures and their regularization cannot be recognized. The fe_sphere graph as the most regular graph, but also the power-law based cond-mat graph both have a very high regularization with $C = 0.1$ while being completely different in structure. Contrary delaunay_n13 and delaunay_n14,

two graphs that originate from the same generative model, have different regularization, with the regularization of delaunay_n14 being more relaxed. We expect this variance in regularization to come from the natural order of node IDs, which determines the streaming order and therefore also the prediction outcome. The more noisy the prediction and the less homogeneous the block assignments, the more regularization is probably necessary to prevent overfitting. This supports the hypothesis, that the choice of hyperparameters cannot be generalized easily, but is different from one graph and its characteristics to another.

GBDTs. Tuning the hyperparameters of the GBDTs model is based on three stages. First, we tune the trade-off between the number of weak classifiers $n\_estimators$ and the learning rate η. Then we tune tree-specific parameters, i.e., the maximum tree depth and the subsampled percentage of nodes to train each weak classifier on. Then the regularization parameter λ is optimized.

We first fix the maximum tree depth to three, the subsampling to 50%, and λ to one. A grid search is performed for $n\_estimators \in \{100, 200, 300, 400, 500\}$ and $\eta \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$. Based on its results we sequentially first search through the best $max\_depth \in \{1, 2, 3, 4, 5, 6\}$, then $subsample \in \{0.5, 0.6, 0.7, 0.8, 1.0\}$ and last but not least $\lambda \in \{10^i | -3 \leq i \leq 1\}$. After these tuning experiments, we get the following final settings:

Graph	**#Estimators**	η	**max_depth**	**subsample**	λ
delaunay_n13	100	0.05	1	0.5	0.1
delaunay_n14	100	0.05	1	0.6	0.01
astro-ph	300	0.2	1	1	10
cond-mat	400	0.05	1	0.7	10
fe_4elt2	100	0.15	1	0.6	1
fe_sphere	400	0.1	1	0.5	1
road-minnesota	100	0.05	1	0.5	1
road-euroroad	100	0.05	4	0.7	0.001
email-EuAll	500	0.05	2	0.5	0.001
wordassociation-2011	200	0.05	1	1	0.1

Table 5.2: GBDTs hyperparameter evaluation: #Estimators, η, max_depth, subsample and λ

Once again, the choice of hyperparameters cannot be generalized easily. First of all, we recognize that the intuitive assumption that more estimators lead to lower learning rates does not hold true. The graph astro-ph even shows the complete opposite having the highest learning rate among all graph instances and yet having 300 estimators considered best. Most graphs nevertheless seem to need decision stumps as their weak classifier, with the exception being the scale-free email-EuAll graph with a maximum depth of two and road-euroroad with a maximum depth of even four. Regarding the subsample of nodes taken to train each classifier, the percentage differs from graph to graph, for most graphs between 50% and 70% of the nodes are used for training. The highest subsampling is used for the two power-law based graphs astro-ph and wordassociation-2011, which do not subsample at all but take all nodes available for training each weak classifier. The regularization parameter λ is chosen highest for the artificial and social graphs, surprisingly also for the road-euroroad graph, and remains at a moderate level of one for almost all other graphs.

Support Vector Machine. Besides the kernel to use and the kernel-specific parameters to choose like the degree of the polynomial kernel or the inverse of the radius of influence of support vectors γ from the RBF kernel, one also needs to evaluate the inverse of the regularization strength C. Furthermore, specific to scikit-learn, the polynomial kernel is implemented as $K(x_i, x_j) = (\gamma x_i^T x_j + 1)^p$ with an additional scaling parameter γ that needs to be determined. Specific to the SVM, we cut the training dataset once the 10 000 node mark is reached as the SVM's time to fit the training data grows quadratically. Still, we make sure to have a class-balanced dataset for the following experiments.

We quickly figure out that letting PyTorch decide, which γ to use, is the best choice, the default configuration scales it as $1/(\#features \cdot Var(X))$.

The next step is to find out which kernel to use. Therefore we fix C to 0.01 and run a grid search, the default degree of the polynomial kernel is set to three. The last grid search runs to find the optimal parameter value for C. We specify its search space by the set of $\{10^i | -3 \leq i \leq 1\}$. The experiments give the following result:

Graph	**Kernel**	**Degree**	**C**
delaunay_n13	rbf	1	1
delaunay_n14	rbf	1	10
astro-ph	rbf	1	10
cond-mat	rbf	1	1
fe_4elt2	rbf	1	1
fe_sphere	rbf	1	0.1
road-minnesota	rbf	1	0.1
road-euroroad	rbf	1	1
email-EuAll	rbf	1	1
wordassociation-2011	rbf	1	10

Table 5.3: SVM hyperparameter evaluation: Kernel, degree and regularization C

We see that all graphs can be best partitioned using the RBF kernel. We know, that the RBF kernel is capable to fit a much larger function space than any polynomial kernel would be capable of, for the cost of data and fitting time necessary. As it is therefore more flexible than a restricted polynomial kernel, the reported results are intuitive. Nevertheless, different regularization terms show, that due to each graph's topology, no common regularization strength can be defined. Because of its completely regular structure, the graph fe_sphere was expected to have a lower regularization, still, the grid search returns $C = 0.1$ which indicates higher regularization. On the other hand, complex graphs like astro-ph or wordassociation-2011 have really low regularization with $C = 10$, where it would be intuitive to have a higher regularization. This supports the hypothesis, that the choice of hyperparameters cannot be generalized easily.

Side notes. We also experimented with the hyperparameter selection for $k = 2$, which leads to different configurations that are still graph-individual. An interesting discovery on the other hand was that normalized features led to more homogeneous regularization factors than non-normalized features for the parametric models. One could therefore say, that the more complex the domain (measured by k and the activation of feature normalization), the more individual the hyperparameters get in order to fit the domain closely.

To summarize, we recognize all graphs have their own specific best hyperparameter choices and that no general patterns can be derived. We therefore conclude that the choice of hyperparameters depends a lot on the individual graph's topology, its natural order of nodes, and internal structure. We store the above configuration and reuse it for the following experiments while each time loading a graph's individual hyperparameters.

5.4.4 Balance Heuristics

A comparison between the two balance heuristics introduced can be found in the table Table 5.4. We recognize a strong tendency that the MostProbable heuristic outperforms the LeastLoaded heuristic. This stands in contrast to the findings of Abbas [12], who reports the opposite but for the edge partitioning scenario on attributed graphs. We will use the MostProbable heuristic for all future experiments.

	Graph	Baseline	GBDTs	GraphSAGE	Partitioner
MostProbable	delaunay_n13	**4 381**	**4 152**	**4 116.4**	**4 198.6**
	delaunay_n14	**7 989**	**7 361**	**7 323.2**	**7 351.8**
	astro-ph	**39 642**	39 251	39 744.8	40 088.4
	cond-mat	**11 348**	11 737	11 691.2	11 657.8
	fe_4elt2	**6 115**	6 488	**6 367.4**	**6 432.8**
	fe_sphere	3 885	**3 436**	**3 407.8**	**3 629**
	road-minnesota	**387**	**369**	**386.6**	**419.8**
	road-euroroad	**260**	269	**259.4**	**261.8**
	email-EuAll	27 662	25 027	26 987.8	28 045
	wordassociation-2011	**36 168**	**36 392**	**36 091.6**	**36 312.2**
	Geometric Mean	**5 722.45**	**5 562.83**	**5 588.05**	**5 723.00**
LeastLoaded	delaunay_n13	4 381	4 156	4 147.2	4 722.8
	delaunay_n14	8 000	7 561	7 374.8	8 270.8
	astro-ph	39 657	**39 244**	**39 640.6**	**39 679**
	cond-mat	11 361	**11 475**	**11 546.4**	**11 569.2**
	fe_4elt2	6 374	**6 312**	6 616	6 653.6
	fe_sphere	**3 839**	3 526	3 701.2	3 818.8
	road-minnesota	388	486	398.6	512.8
	road-euroroad	260	**267**	260.4	271.2
	email-EuAll	**27 575**	**24 989**	**26 420.6**	**27 469.8**
	wordassociation-2011	36 252	37 296	36 398.6	36 595
	Geometric Mean	5 742.07	5 729.00	5 667.92	6 031.30

Table 5.4: Balance heuristic evaluation: Total cut sizes by balance heuristic. Experiments run on the tuning dataset with $k = 16$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

5.4.5 Streaming Mode

When tuning the streaming mode, there are three parameters to empirically evaluate. In the following, we will investigate the buffer size b, the step size s, and the number of prediction propagation rounds.

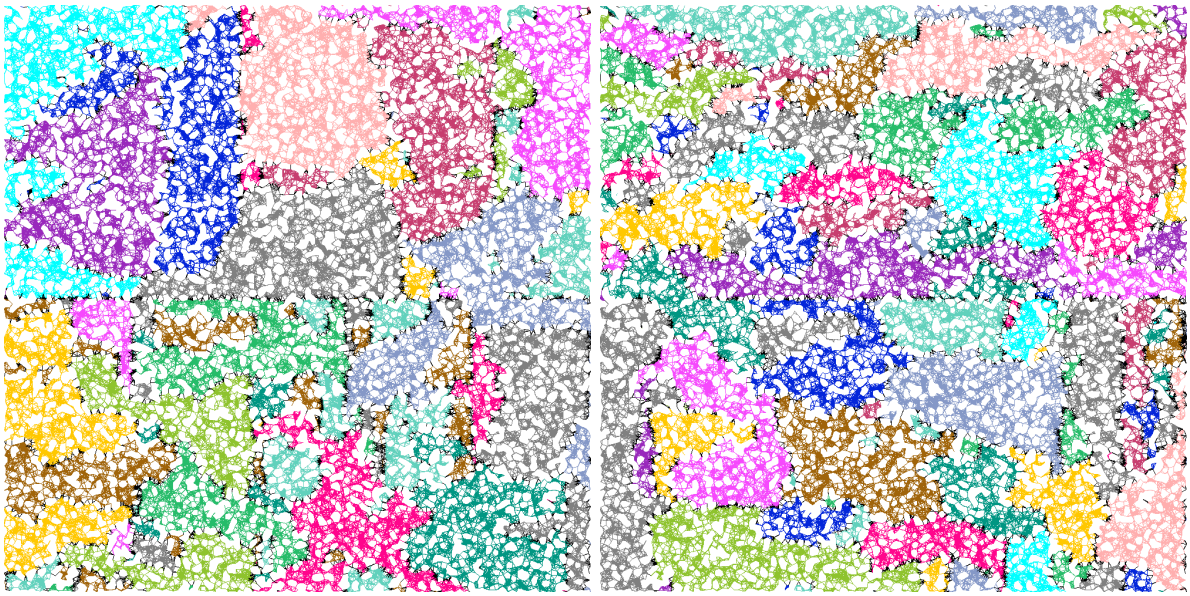
An investigation of the impact of the buffer size on the partitioning quality for the Partitioner model can be found in Table 5.5. First, we recognize that increasing the buffer size brings a performance boost as a more global view is given when building the feature vectors and predicting the block assignments. This performance boost lasts until a buffer size of $b = 256$ when the partitioning quality starts to decrease again.

Graph	**16**	**32**	**64**	**128**
delaunay_n13	4 261	**4 111.2**	4 213	4 153.4
delaunay_n14	7 633.2	7 386.6	7 239	7 443.6
astro-ph	40 432.6	40 136.2	39 965.6	40 363.4
cond-mat	11 776	11 734.4	11 691.6	11 677.8
fe_4elt2	6 509	6 512.4	6 373.4	6 373.6
fe_sphere	3 778.2	3 599.4	3 656.2	3 520.4
email-EuAll	28 186.6	28 187.4	28 173	28 146.4
wordassociation-2011	36 534	**36 032.8**	36 252.2	36 063.4
Geometric Mean	11 862.76	11 655.12	11 649.30	11 617.84
Graph	**256**	**512**	**1024**	**100%**
delaunay_n13	4 149.4	4 153.6	4 182.8	4 706
delaunay_n14	7 318.4	**7 075.2**	7 456.4	8 525.6
astro-ph	40 145	40 278	**39 910.8**	40 333.6
cond-mat	11 653.8	**11 647.2**	11 656	11 877.6
fe_4elt2	6 208.4	6 389.8	**6 197.4**	6 723.6
fe_sphere	**3 516.4**	3 573.4	3 556.6	3 957.4
email-EuAll	28 191	28 176	28 223.8	**27 489.4**
wordassociation-2011	36 100	36 174.4	36 431.6	36 571
Geometric Mean	**11 545.15**	11 568.79	11 604.38	12 271.17

Table 5.5: Buffer size evaluation: Total cut sizes by buffer size for the Partitioner model. Experiments run on the tuning dataset with $k = 16$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Note that the road networks were not included in this experiment because of their limited number of nodes, which would not have allowed such big buffers for the test set. Also, we performed an exhaustive prediction propagation exclusively for this experiment to make

sure the benefits of bigger buffers can actually be utilized. To explain why the total cut sizes increase again after $b = 256$, we have a look at the partition plots of the evaluation graph rgg_n_2_15_s0 in Figure 5.3, which provides node coordinates for drawing:



(a) $b = 256$ (b) $b = 100\%$

Figure 5.3: Buffer size evaluation: Partition visualization by buffer size for the Partitioner model on the rgg_n_2_15_s0 graph

Typical for streaming graph partitioning, we recognize that the blocks are not always a single unit but rather clustered across the graph. These clusters are slightly smaller and seemingly more frequent when the buffer size is chosen to match the size of the graph, which causes the higher total cut size. A real explanation for this phenomenon cannot be given. We choose a buffer size of 256 for all remaining experiments.

The step size is crucial for the duration of the node stream. To empirically investigate the impact of the step size on the streaming time, we slightly change the experimental setup. We now train on the entire email-EuAll graph and store the model parameters instead of training on only 80% of the nodes. We then take the same graph and predict all block labels. This way we have a larger number of nodes to run the prediction. Note that we are only interested in the time needed for streaming and that therefore the quality improvement that comes from training and predicting on the same graph is irrelevant.

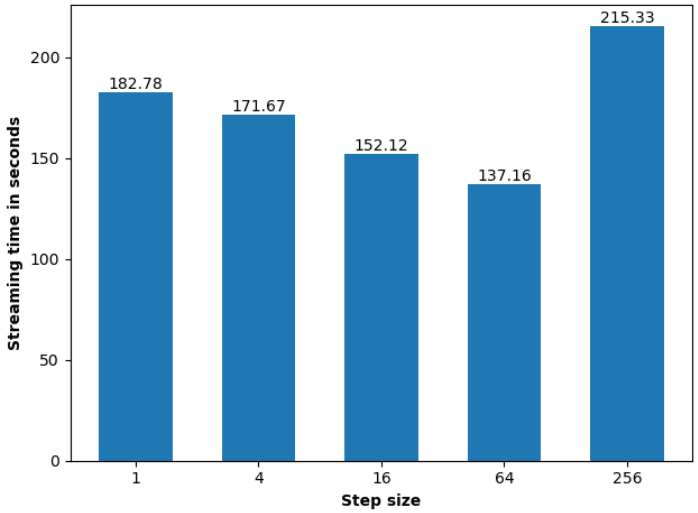


Figure 5.4: Step size evaluation: Streaming time in seconds by step size for the Partitioner model on the email-EuAll graph

We see in Figure 5.4 that the bigger the step size the faster the streaming and prediction routine, except for step size $s = 256$ where the step size equals the buffer size. We explain this anomaly as the streaming time being dependent on both the step size and the extent of prediction propagation to be performed for individual nodes. The bigger the step size, the more unclassified nodes will be in the buffer, such that the additional neighborhood feature recalculation and possibly repredictions neutralizes the decreasing effect of the step size on the streaming time.

On the other hand, the bigger the step size, the noisier the prediction as subsequent buffers are seen more and more separate when the overlap shrinks. This results in more restricted views on the overall graph, which impacts the partitioning quality as shown in Table 5.6:

Graph	**1**	**4**	**16**	**64**	**256**
delaunay_n13	**4 111.2**	4 260	4 198.6	4 222.6	4 298
delaunay_n14	7 422.8	**7 319.6**	7 351.8	7 496.8	7 852.2
astro-ph	**39 732.8**	40 120.4	40 088.4	40 056.8	40 184.6
cond-mat	11 668.6	11 679.8	**11 657.8**	11 664.4	11 667.4
fe_4elt2	**6 369**	6 371.6	6 432.8	6 448.4	6 581
fe_sphere	**3 450.8**	3 575.4	3 629	3 597.2	4 456
email-EuAll	27 972.6	28 135.4	28 045	**27 803.6**	28 057.6
wordassociation-2011	**36 216.4**	36 270.6	36 312.2	36 303.4	36 473.2
Geometric Mean	**11 542.28**	11 651.58	11 665.51	11 679.72	12 149.80

Table 5.6: Step size evaluation: Total cut sizes by step size for the Partitioner model. Experiments run on the tuning dataset with $k = 16$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

The worst performance is achieved in case the step size is set equal to the buffer size, here $s = 256$, whereas smaller step sizes like $s = 1$ perform better. We choose a step size of 16 for all future experiments to achieve a good quality / running time trade-off.

When increasing the prediction propagation rounds, we recognize that more such rounds improve the partitioning quality. In Table 5.7 having a single prediction round gives worse results than increasing the prediction propagation rounds to five.

Graph	1 round	2 rounds	3 rounds	4 rounds	5 rounds
delaunay_n13	4 388.4	4 155.4	4 116.4	**4 045.6**	4 047.6
delaunay_n14	7 801.6	7 331.2	7 323.2	7 269.2	**7 241.8**
astro-ph	40 658.6	39 820.4	**39 744.8**	40 263.4	39 948
cond-mat	11 726	**11 648**	11 691.2	11 718.2	11 694.6
fe_4elt2	6 601	6 375.2	**6 367.4**	6 454.4	6 438.2
fe_sphere	4 757.2	**3 349.8**	3 407.8	3 526.6	3 487.4
road-minnesota	406.8	382.4	386.6	**367.6**	**373.8**
road-euroroad	270.6	**258.2**	259.4	258.4	259.6
email-EuAll	27 075.8	27 248.8	26 987.8	27 439.4	**26 952**
wordassociation-2011	36 979.4	36 360.8	36 091.6	36 093.6	**36 020.4**
Geometric Mean	5 959.40	5 584.85	5 588.06	5 588.47	**5 574.26**

Table 5.7: Prediction propagation evaluation: Total cut sizes by prediction propagation rounds performed for the GraphSAGE model. Experiments run on the tuning dataset with $k = 16$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Note that after some threshold the predictions have converged to their final settlement and therefore more prediction propagation rounds do not improve the total cut size result anymore. Also, we see that there is only a slight variance between the reported total cut size results between two to five prediction propagation rounds, which is caused by different random seeds being used for each experiment. Therefore having only three prediction propagation rounds should be enough to propagate the final block assignment predictions throughout the graph, which is why we choose this configuration for all remaining experiments.

We furthermore experiment on the importance of the natural order of nodes. We assume that the more the natural order of nodes resembles a BFS order, the better the features can be generated and the more accurate the prediction will be. This can be proved empirically in Table 5.8 showing far lower total cut sizes when applying a BFS ordering to the node stream beforehand. Nevertheless, as we cannot enforce such a BFS order in a given graph that is streamed from the hard disk drive or from a graph in an online algorithm setting, we will execute all following experiments without additional BFS preprocessing.

	Graph	Baseline	GBDTs	GraphSAGE	Partitioner
Natural Order	delaunay_n13	4 381	4 152	4 116.4	4 198.6
	delaunay_n14	7 989	7 361	7 323.2	7 351.8
	astro-ph	39 642	39 251	39 744.8	40 088.4
	cond-mat	11 348	11 737	11 691.2	11 657.8
	fe_4elt2	6 115	6 488	6 367.4	6 432.8
	fe_sphere	3 885	**3 436**	**3 407.8**	3 629
	road-minnesota	**387**	**369**	**386.6**	**419.8**
	road-euroroad	260	269	259.4	261.8
	email-EuAll	27 662	25 027	26 987.8	28 045
	wordassociation-2011	36 168	36 392	36 091.6	36 312.2
	Geometric Mean	5 722.45	5 562.83	5 588.05	5 723.00
BFS Order	delaunay_n13	**3 172**	**3 207**	**3 286.2**	**3 590.6**
	delaunay_n14	**5 621**	**5 117**	**5 798.6**	**6 229.6**
	astro-ph	**31 302**	**32 046**	**32 120.2**	**32 142.6**
	cond-mat	**8 372**	**8 579**	**8 736.8**	**8 819.6**
	fe_4elt2	**3 035**	**3 178**	**3 255.4**	**3 373.4**
	fe_sphere	**3 730**	3 483	3 448.8	**3 583.4**
	road-minnesota	409	388	394.8	439.6
	road-euroroad	**203**	**211**	**211.4**	**214.4**
	email-EuAll	**23 231**	**23 341**	**23 289.2**	**23 521.8**
	wordassociation-2011	**33 373**	**33 348**	**33 408.8**	**33 374.8**
	Geometric Mean	**4 499.97**	**4 469.65**	**4 561.33**	**4 735.54**

Table 5.8: Node order evaluation: Total cut sizes by node order. Experiments run on the tuning dataset with $k = 16$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

5.5 Evaluation Phase

5.5.1 Generalization

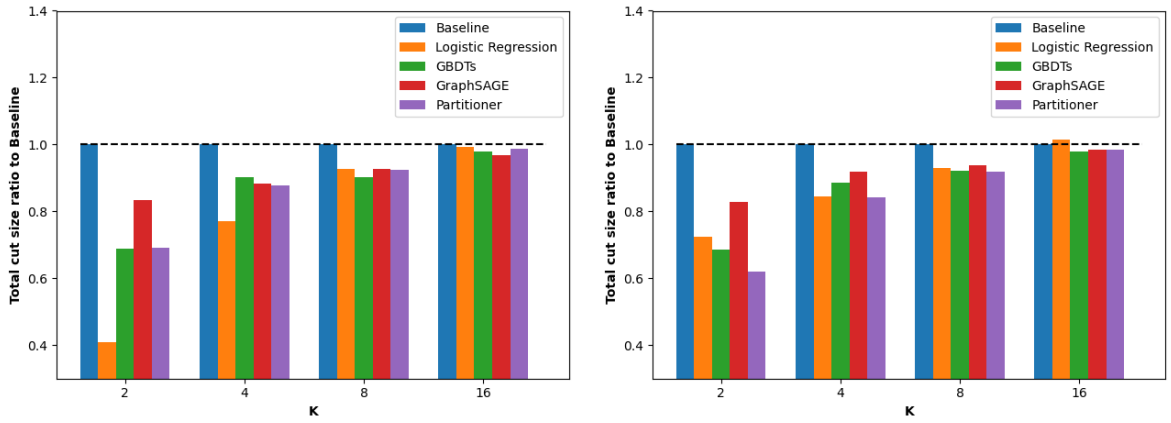
The previous experiments explored which features to use, which balance heuristic works best and how to configure the streaming mode. Now that we have all the tuning done, we can perform a final series of experiments on the separate evaluation dataset to explore the generalization capabilities of the models. To do this, we construct two generalization scenarios:

1. **Instance**: We uniformly split every graph instance by 80% of the nodes for training and 20% of the nodes for prediction. Also, the reported total cut sizes are constructed by using the optimal block labels for the 80% training nodes and the predicted labels for the rest. The experiments are averaged over five runs with different random seeds for each graph instance.
2. **Group**: For this experiment, we would like to evaluate the hypothesis, whether the prediction works well on unseen, structurally similar graphs. Therefore we limit the training and prediction to equal groups of graphs. We train on all graphs from the same group in the tuning dataset and predict the partitions of the graphs in the equivalent group in the evaluation dataset. The experiments are averaged over five runs with different random seeds for each graph instance.

As we experience faster convergence than expected for the deep learning models, we lower the number of training epochs from 2 500 to 1 000.

Instance generalization.

The instance generalization experiments are run for $k \in \{2, 4, 8, 16\}$ blocks and $\epsilon \in \{0.0, 0.03\}$ imbalance. The detailed results are reported in Appendix C. The following analysis excludes the SVM model as it struggled to keep up with its competitor models, because of having an upper bound on the data to be used for training. This threshold was set to 10 000 training instances and turned out to be a major limiting factor for the performance of the SVM model, but else the training time would have exploded because of its quadratic scaling factor.



(a) $\epsilon = 0.0$ (b) $\epsilon = 0.03$

Figure 5.5: Instance generalization evaluation: Total cut size ratio between the selected model and the Fennel based baseline algorithm by number of blocks averaged over all graphs

We now first would like to investigate the overall average performance of the models over the different values of k in Figure 5.5. We set the geometric mean over the total cut sizes in relation to the Fennel based baseline algorithm and recognize that the larger k gets, the more all models approach the performance of the Fennel based baseline algorithm.

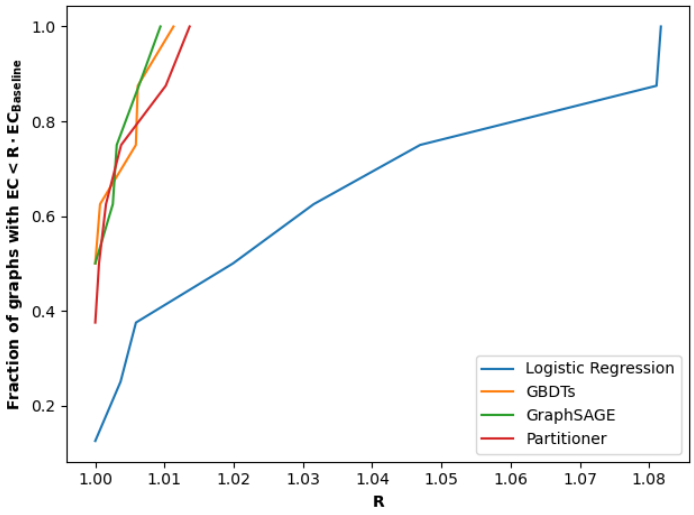
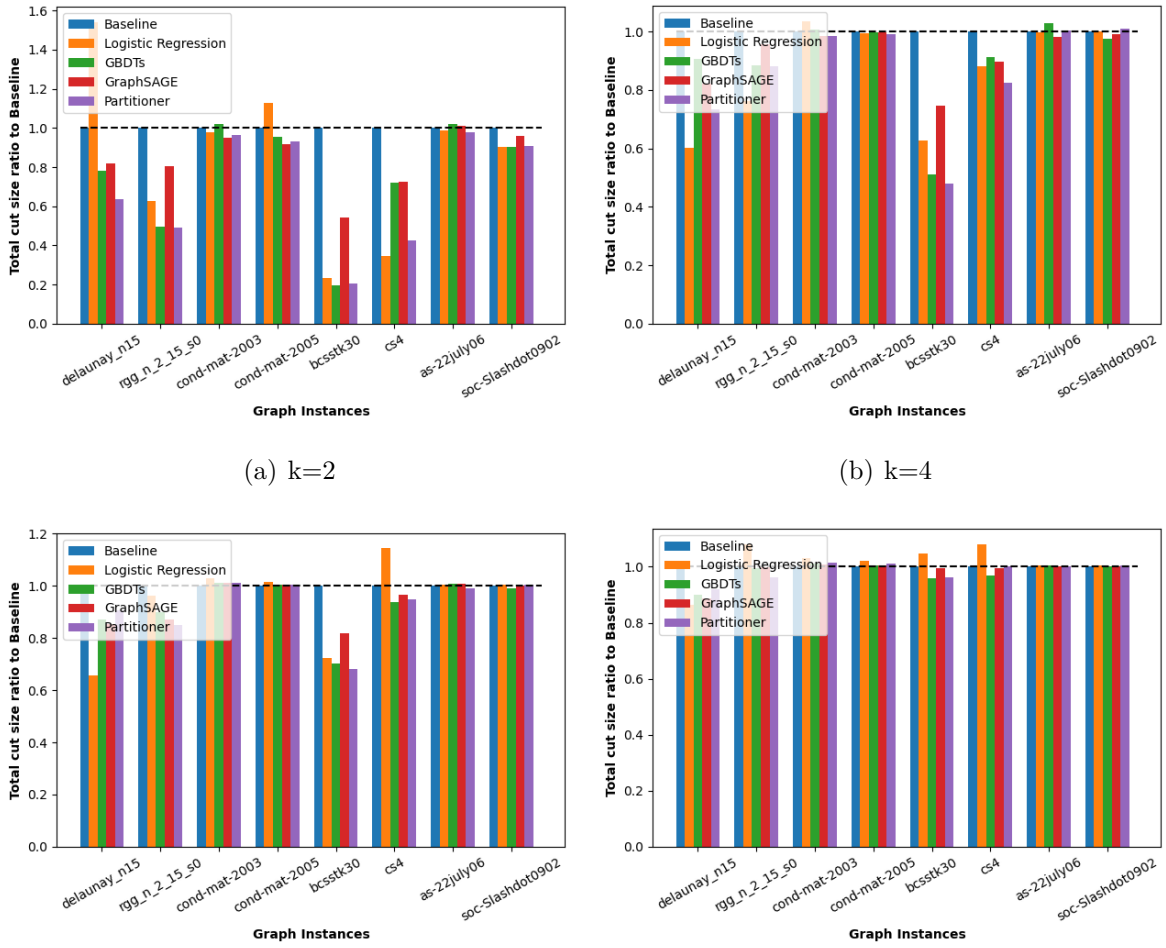


Figure 5.6: Instance generalization evaluation: Performance profile for $k = 16$ and $\epsilon = 0.03$ compared to the Fennel based baseline algorithm

Looking at the performance profile in Figure 5.6, it is hard to determine a clear winner between the GBDTs model and the deep learning models. For high values of k, the Logistic Regression model is outperformed by the other models, whereas for lower values of k the Logistic Regression model might still be the best choice as seen for the perfectly balanced bisection setting. Here the Logistic Regression classifier manages to be just maximally 1.1 times worse than the Fennel based baseline algorithm compared to the Partitioner model being approximately 1.3 times worse than the Fennel based baseline algorithm in the worst case. After a more detailed investigation, the winners are the GraphSAGE and XGBoost model as not less than 50% of the graph instances are partitioned better than with the Fennel based baseline algorithm and maximally 1.015 times worse than with the Fennel based baseline algorithm. The Partitioner model might be up to 1.3 times worse than the Fennel based baseline algorithm for lower values of k and $\epsilon = 0.0$ imbalance, but else has similar performance. This means that, for such graph instances that are partitioned worse than with the Fennel based baseline algorithm, GraphSAGE alone is sufficient to make good predictions and that the additional partitioning module (a densely connected network) that the Partitioner model brings does not add any benefits.



(a) k=2

(b) k=4

(c) k=8

(d) k=16

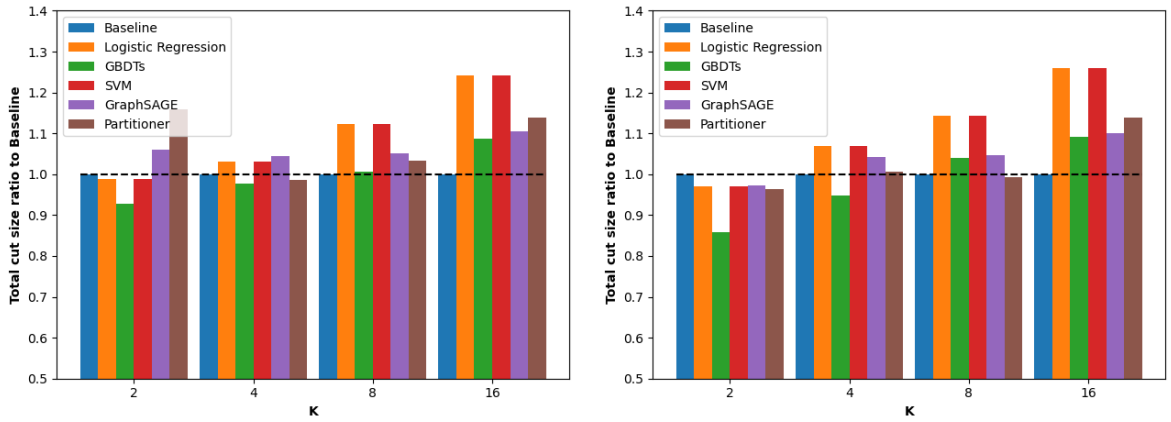
Figure 5.7: Instance generalization evaluation: Total cut sizes by evaluation graph instance normalized to the total cut size provided by the Fennel based baseline algorithm for $\epsilon = 0.03$

In the more detailed total cut size plots in Figure 5.7, we can also confirm that the more complex the problem becomes and the higher k, the more the GBDTs, GraphSAGE and Partitioner model seem to be the best choice.

To summarize, in case the task of streaming graph partitioning is limited to evolving graph settings and one only focuses on partitioning quality, low values of k imply no recommendation to use a specific model while for higher values of k starting from $k = 8$ a deep learning model or GBDTs are the best choice.

Group generalization.

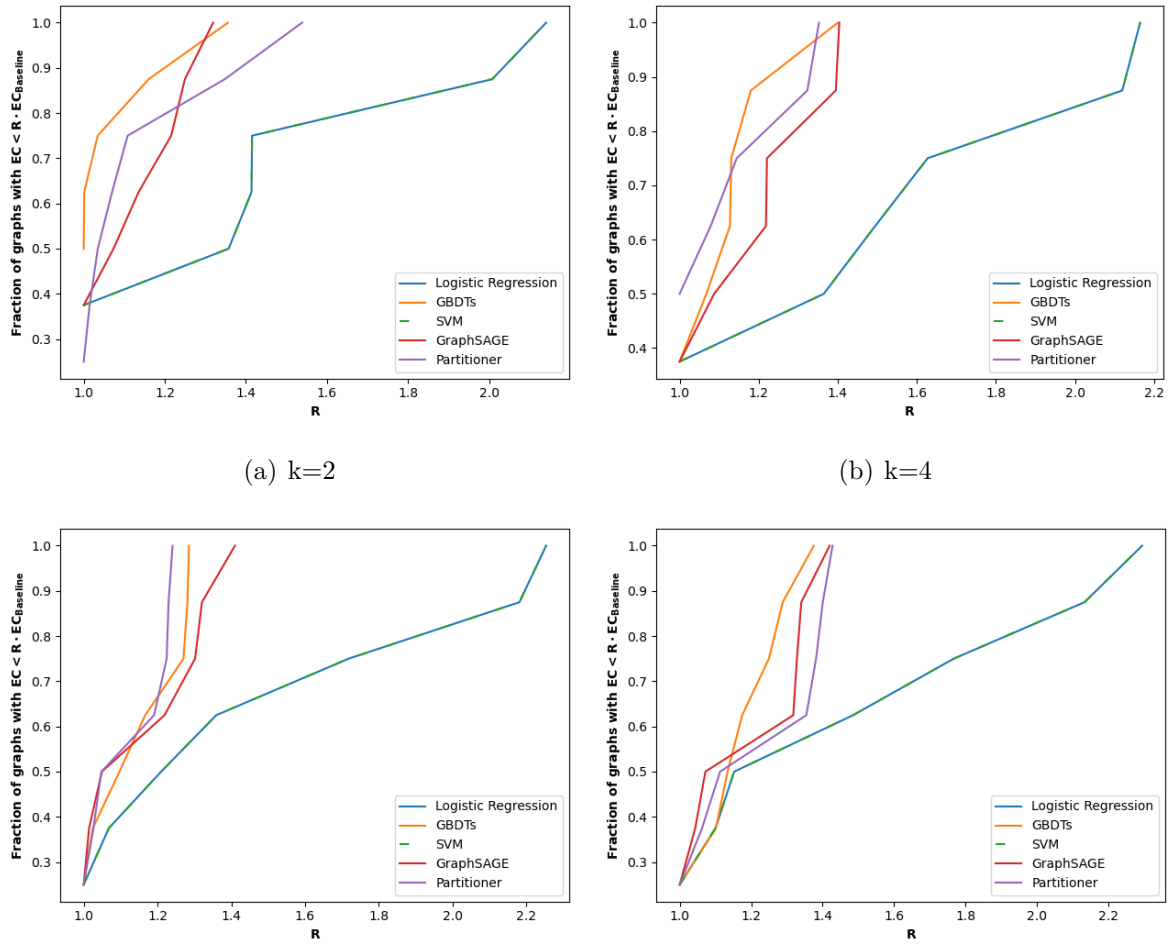
We run the group generalization experiments for all numbers of block $k \in \{2, 4, 8, 16\}$ and $\epsilon \in \{0.0, 0.03\}$ imbalance. When investigating the geometric mean of the total cut sizes over all graph instances, put into ratio with the Fennel based baseline algorithm, we obtain Figure 5.8. The exact results are reported in Appendix C.



(a) $\epsilon = 0.0$ (b) $\epsilon = 0.03$

Figure 5.8: Group generalization evaluation: Total cut size ratio between the selected model and the Fennel based baseline algorithm by number of blocks averaged over all graphs

We recognize that starting from $k = 8$, almost no model can outperform the Fennel based baseline algorithm anymore. Among all models, the GBDTs report the best results on average. The parametric models, i.e., the Logistic Regression classifier and the SVM, can keep up well with the deep learning models for $k = 2$ and even outperform them in the perfectly balanced case. Note that both of them report the same total cut sizes as their decision boundaries are very similar due to the feature normalization. For bigger values of k, the deep learning models notably achieve lower total cut sizes on average compared to the parametric models, but no strict order in performance can be determined between them.



(a) k=2

(b) k=4

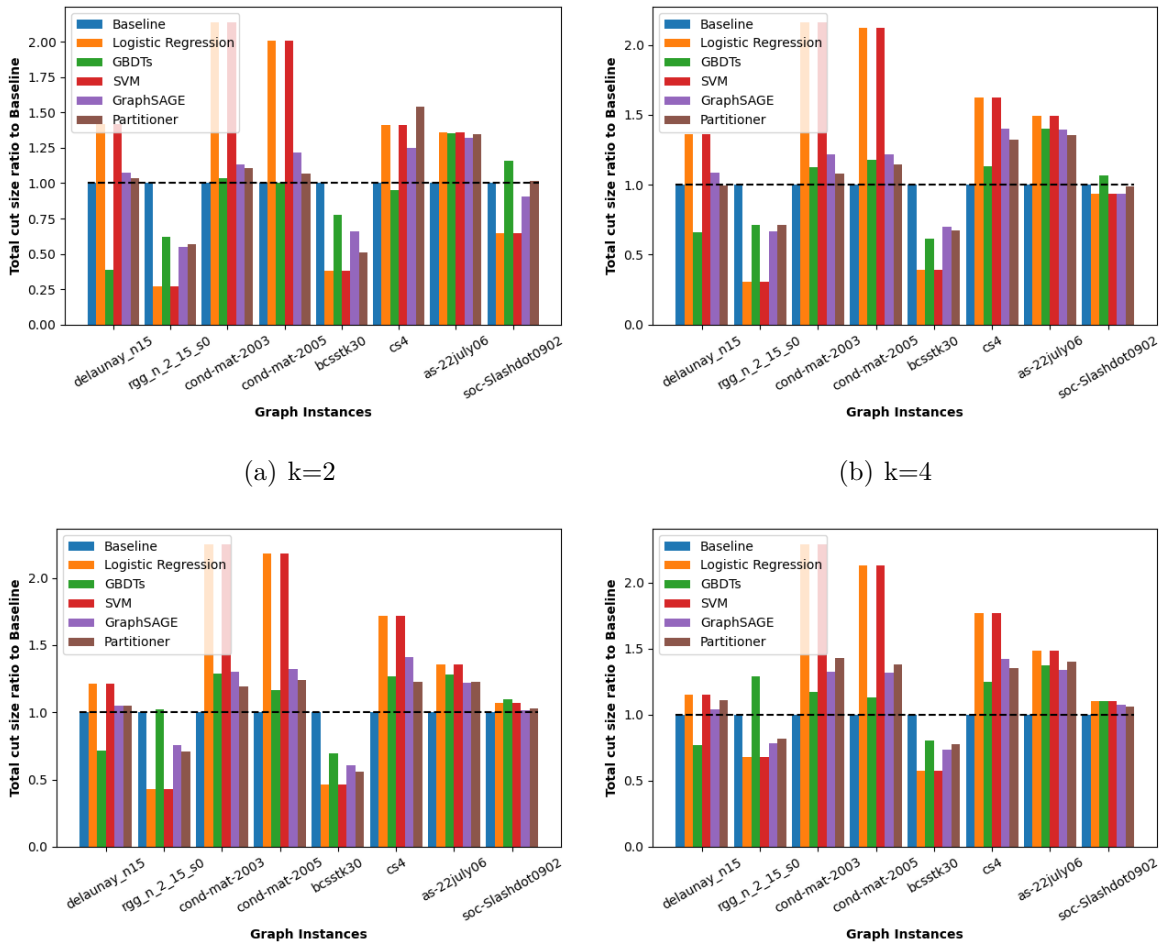
(c) k=8

(d) k=16

Figure 5.9: Group generalization evaluation: Performance profiles by k for $\epsilon = 0.03$ compared to the Fennel based baseline algorithm

By looking at the performance profile plots in Figure 5.9, we see that the curve of XGBoost is the fastest growing not surpassing $R = 1.5$. The deep learning models come next that only surpass $R = 1.5$ for $k = 2$. A clear winner between the GraphSAGE model and the Partitioner model cannot be determined. The worst performance have the parametric models Logistic Regression and SVM that can be up to 2.5 times worse than the Fennel based baseline algorithm. We also recognize that compared to instance generalization the factor R becomes much higher for group generalization.

Generalization to unseen graphs works best in case the graphs used for training are very close in structure to the graphs used during prediction, having the same type of degree distribution alone is not sufficient to guarantee generalization. This is the reason why on average the Fennel based baseline algorithm still performs best, especially for higher k. If we can ensure an equal generative model behind graph instances, generalization to unseen graphs indeed works out. This can best be seen in the delaunay_n15 graph. It was generated using KaGen[3] [63], just as the delaunay graphs used during training.



(a) k=2

(b) k=4

(c) k=8

(d) k=16

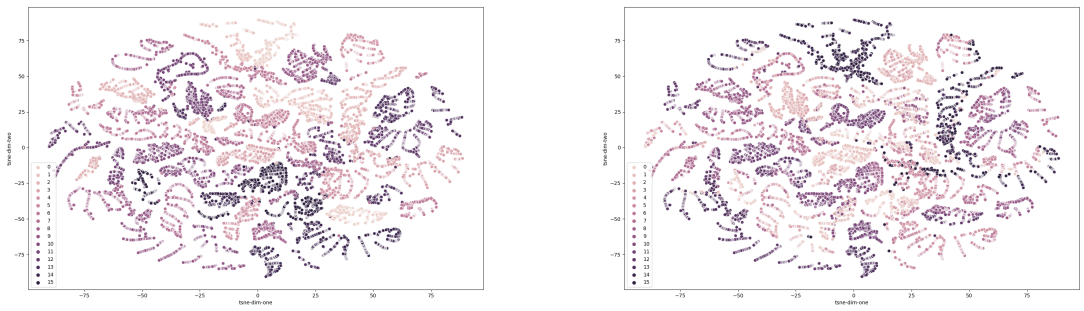
Figure 5.10: Group generalization evaluation: Total cut sizes by evaluation graph instance normalized to the total cut size provided by the Fennel based baseline algorithm for $\epsilon = 0.03$

[3]https://github.com/sebalamm/KaGen

Looking at Figure 5.10, we see that delaunay_n15 is always best partitioned using the GBDTs, which confirms generalization capabilities for unseen graphs. One only needs to make sure that the generative model behind the graph is very similar to the one behind the training graphs. This matches the findings in related work like GAP [10] and GCNSplit [12, 13]. GCNSplit also reports decreasing partitioning performance once the evolving graphs develop further away from the training graphs.

The hypothesis, that the prediction works well on unseen, structurally equivalent graphs, can therefore be proved empirically but needs a very strict definition of structural equivalence. Having a similar degree distribution is not enough to make a generalization to unseen graphs possible, small changes, e.g., in the degree exponents can lead to a huge difference. Instead, the generative process behind graphs needs to be the same as seen for the delaunay_n15 graph.

As a second example besides the GBDTs on the delaunay_n15 graph, also the GraphSAGE model consistently performs better on the rgg_n_2_15_s0 graph. Already in the feature space in Figure 5.11, we can see, that the generated Fennel score based features after prediction propagation lead to a clustered feature space, that enables us to make good predictions. When comparing the feature space labeled by the predicted block assignments with the same feature space labeled by the true block assignments, we see that the prediction comes close to what was expected.

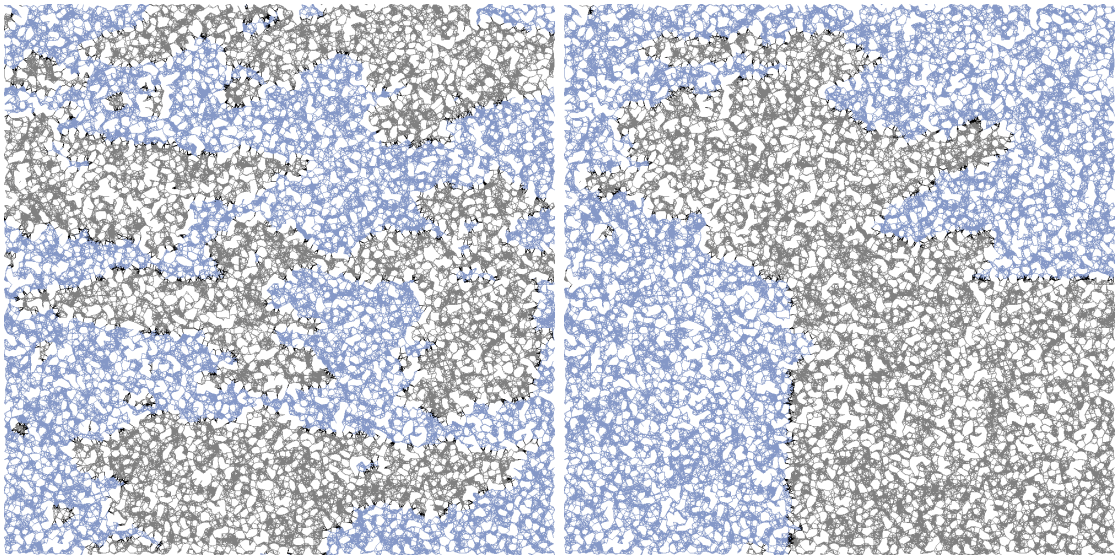


(a) Labeled by predicted block assignments

(b) Labeled by true block assignments

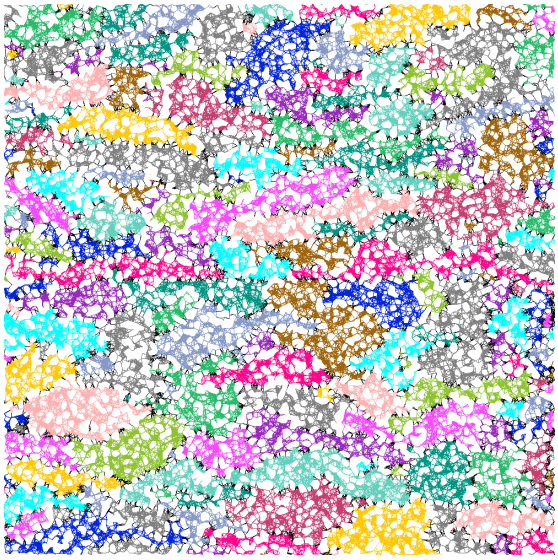
Figure 5.11: Group generalization evaluation: T-SNE feature visualization of the generated features during prediction of the GraphSAGE model for $k = 16$ and for $\epsilon = 0.03$

As the prediction comes close to what an offline graph partitioner like KaFFPa [14] would do, this might lead to the assumption that the supervised learning approach has an influence on why the GraphSAGE model and also other models can correct the errors of the Fennel heuristic. The effect of correcting the errors of the Fennel heuristic becomes especially clear when investigating the partition visualization in Figure 5.12.

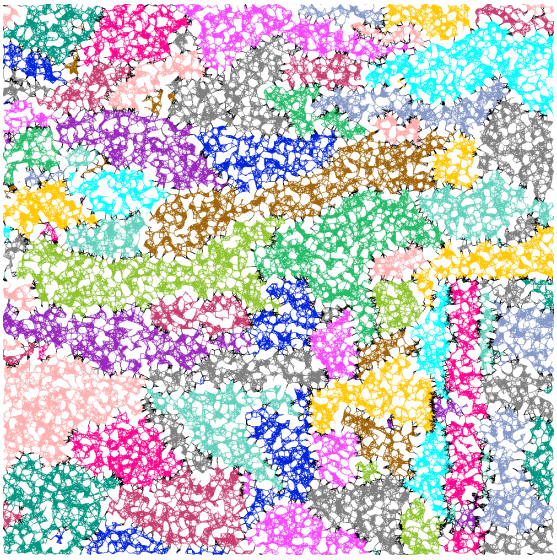


(a) Baseline ($k = 2$)

(b) GraphSAGE ($k = 2$)



(c) Baseline ($k = 16$)



(d) GBDTs ($k = 16$)

Figure 5.12: Group generalization evaluation: Generated partitions from the GraphSAGE model for rgg_n_2_15_s0 and $\epsilon = 0.03$

While the Fennel based baseline algorithm tends to produce clustered partitions, this effect more and more vanishes for the GraphSAGE model the simpler the problem becomes, i.e., the smaller k becomes. Similar behavior can be observed for the other models.

To summarize, assuming the streamed graph comes from the same generative model as the training graphs, our models seem to enable group generalization capabilities very well as seen by the two models GBDTs and GraphSAGE. The GBDTs provide the lowest total cut sizes on average and should therefore be the model of choice in such settings.

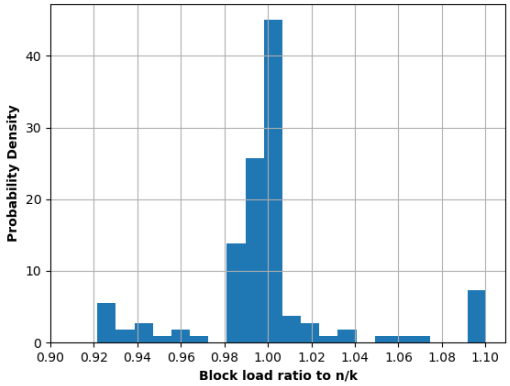
5.5.2 Imbalance

We also experiment with different imbalance parameters to find out how big of an impact less strict balance constraints have on the models. We therefore use the same experimental setting as for the group generalization experiments done before but run the experiment for $\epsilon \in \{0.0, 0.03, 0.07, 0.1\}$ imbalance. This results in the following improvement table, the detailed results can be found in Appendix D.

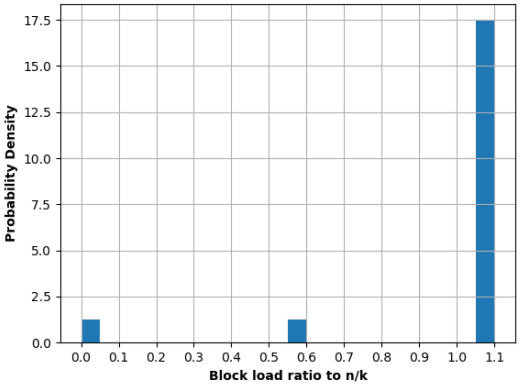
Imbalance	**Baseline**	**Logistic Regression**	**GBDTs**
$\epsilon = 0.03$	0.983	0.997	0.988
$\epsilon = 0.07$	0.978	0.983	0.977
$\epsilon = 0.1$	0.977	0.980	0.975
Imbalance	**SVM**	**GraphSAGE**	**Partitioner**
$\epsilon = 0.03$	0.997	0.979	0.983
$\epsilon = 0.07$	0.983	0.961	0.973
$\epsilon = 0.1$	0.980	0.968	0.963

Table 5.9: Imbalance evaluation: Fraction of cut edges compared to perfectly balanced streaming graph partitioning for $k = 16$

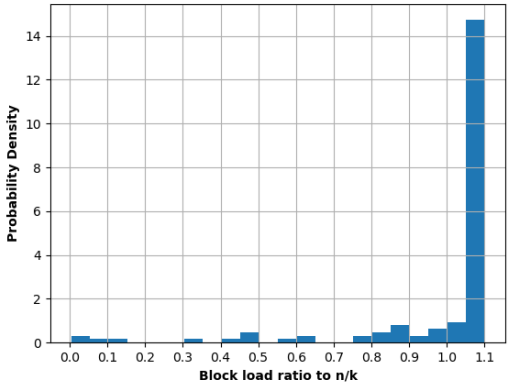
We recognize that a bigger imbalance does not provide a lot of improvement. A 10% increase of imbalance only results in approximately 2% - 4% decrease in total cut size for all models. The biggest improvement is recorded for the Partitioner model with 3.7%.



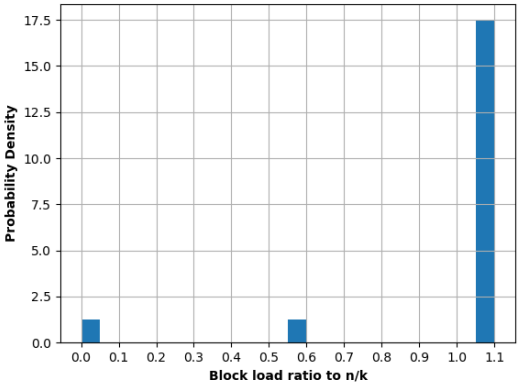
(a) Baseline



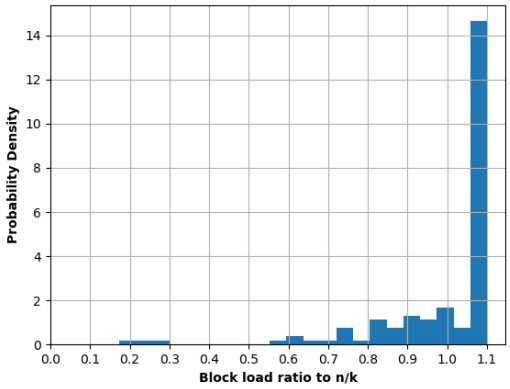
(b) Logistic Regression



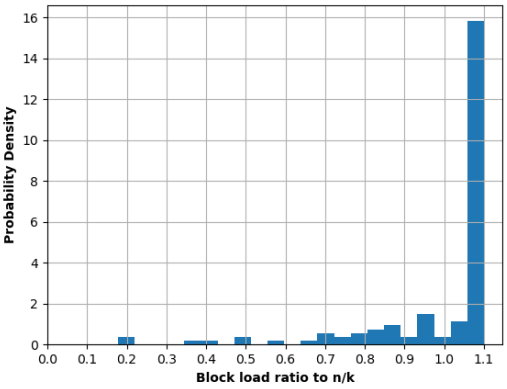
(c) GBDTs



(d) SVM



(e) GraphSAGE



(f) Partitioner

Figure 5.13: Imbalance evaluation: Imbalance distribution aggregated over all graphs for $k = 16$ and $\epsilon = 0.1$

If we would like to investigate, if the additionally allowed imbalance also gets exploited entirely, we make an interesting discovery. Using the group generalization experimental setup with $k = 16$ blocks and $\epsilon = 0.1$ imbalance, we plot the distribution of resulting block load ratios compared to the completely balanced block load $\frac{n}{k}$ over all graphs from the evaluation dataset. We see in Figure 5.13 that the Fennel based baseline algorithm is the only model that still tries to achieve almost perfectly balanced partitioning, even if more imbalance is allowed. The Logistic Regression model and the SVM model achieve very similar distributions and exploit the additionally allowed imbalance maximally, which is the complete opposite of what the Fennel based baseline algorithm does. The GBDTs model as well as the deep learning models GraphSAGE and Partitioner also fully exploit the allowed imbalance, but construct more diversity in the load of blocks.

5.5.3 Running Times

After the extensive series of experiments, we present the running times both for training and prediction for $\epsilon = 0.03$ imbalance in Appendix E. The reported running times include the time needed to generate the features for each node in the streamed graph. On each of the graphs, both training and prediction are performed as partitioning quality is no longer focused. As we come to very similar conclusions for all of the selected graph instances from each group, we choose to use the scale-free as-22july06 graph instance for explanation purposes in this section as it has a more complex structure than the others.

K	**Baseline**	**Logistic Regression**	**GBDTs**
2	- / 4.57	1.60 / 21.31	1.92 / 57.16
4	- / 4.88	1.76 / 22.27	4.58 / 57.40
8	- / 5.12	2.01 / 25.01	3.09 / 58.57
16	- / 5.05	3.00 / 29.52	14.55 / 64.46
Geometric Mean	- / 4.90	2.03 / 24.33	4.46 / 59.33
K	**SVM**	**GraphSAGE**	**Partitioner**
2	4.79 / 23.93	156.81 / 79.45	155.18 / 82.76
4	9.64 / 28.25	162.27 / 97.17	159.74 / 87.22
8	6.72 / 32.16	157.08 / 87.25	158.26 / 92.47
16	9.44 / 39.75	162.66 / 96.42	168.57 / 104.00
Geometric Mean	7.36 / 30.49	159.68 / 89.77	160.36 / 91.28

Table 5.10: Running time evaluation: Running times for training and prediction by model. Experiments run on as-22july06 with $\epsilon = 0.03$. Each experiment is repeated five times using different random seeds. The entries have the format "training time/prediction time" and are measured in seconds.

Looking at Table 5.10, clearly, the deep learning models take the longest both during training and prediction, while not being substantially different from each other. The Logistic Regression classifier being the most simplistic model is also the fastest model on average. While the GBDTs are also rather fast to train on, the prediction time is heavily impacted by the ensemble model character and loses to the SVM and Logistic Regression model. For all models, we see a slight increase in training and prediction time the higher k gets. The biggest increase for all models appears when switching the number of blocks from $k = 8$ to $k = 16$.

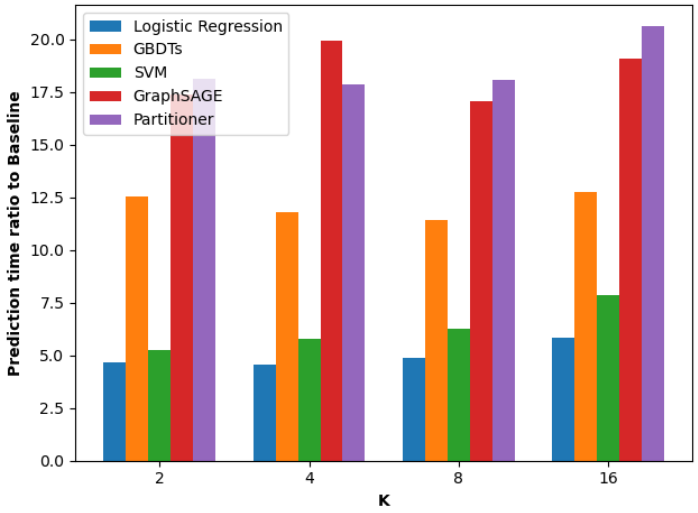


Figure 5.14: Running time evaluation: Ratio of average prediction times between the selected model and the Fennel based baseline algorithm by number of blocks for $\epsilon = 0.03$

Comparing all models to the Fennel based baseline algorithm in Figure 5.14, we see that even the most simplistic model, the Logistic Regression classifier, does approximately need five times the time that the Fennel based baseline algorithm would need for prediction. The GBDTs being one of the best performing models has the third lowest prediction time of all models, while the deep learning based models can even take 20 times as much time as the Fennel based baseline algorithm. This is why in the end we still recommend using the GBDTs model not just from a partitioning quality standpoint, but also regarding its running times.

The reason why all models take notably longer for prediction than the Fennel based baseline algorithm is due to feature normalization, feature standardization, prediction propagation, and the actual inference with the model. If we disable prediction propagation and set the buffer size to one, the effect becomes especially clear as the average time needed for prediction decreases to just 43.13 seconds for the GraphSAGE model and 46.11 seconds for the Partitioner model, so it approximately halves. But as prediction propagation is especially important to get more accurate features and therefore more accurate predictions, this imposes a quality / running time trade-off conflict.

5.5.4 Competitor Comparison

To find out how well our framework performs compared to others, we choose two competitor algorithms. One that is based on a machine learning heuristic and one using a pure algorithmic heuristic.

As, to the best of our knowledge, the only algorithm that applies machine learning to streaming graph partitioning is GCNSplit from Abbas [12] and Zwolak et al. [13], we use this algorithm as the machine learning based competitor. GCNSplit nevertheless takes some different assumptions than we do with our algorithm so far. First, it is an algorithm designed for edge streaming which is why we set the task to edge partitioning in this section. Furthermore, it is applied to attributed graphs, i.e., graphs whose nodes have application domain-specific features assigned to them. The algorithm does not generate features from the pure graph structure as we do with our Fennel score based feature combined with prediction propagation. Also, GCNSplit mainly focuses on unsupervised training by optimizing the continuous relaxation of the expected normalized cut objective with an additive balance term. We train our models supervised using block labels computed with the sophisticated KaFFPa offline multilevel graph partitioner [14]. When being trained unsupervised, GCNSplit predicts the block assignment of an edge by first predicting the block assignment probabilities of both the source and the target node of the edge and then taking the block proposal having the maximum probability in either of both nodes. This way no edge features are required, the model's core predictions are still node based and GCNSplit's training procedure can optimize the node based continuous relaxation of the expected normalized cut objective. This also means our supervised training procedure does not need to be adapted and can continue to use the node based block labels computed by KaFFPa [14]. Luckily, our framework also already supports edge streaming.

The prediction logic is performed in the same way as for GCNSplit. The only problem that arises is building the Fennel score based feature during streaming. Neighboring nodes can no longer be assigned to just a single block but can be replicated to multiple blocks instead. As the original formula to calculate Fennel scores (see Section 2.2) is based on the block assignments of neighboring nodes, which are now not necessarily any longer unique, we modify the Fennel heuristic to use the block assignments of adjacent

edges instead of its adjacent neighboring nodes. We keep the parameters α and γ the same as before.

For the second algorithmic competitor, we decide to compare against 2PS-L, a stateful two-phase streaming algorithm at a linear running time for edge partitioning by Ruben et al. [21]. We shortly outline the algorithm in the following. In the first phase of the algorithm, an extension of the streaming clustering algorithm proposed by Hollocou et al. [64] is used to build a clustering of the entire graph. The extension makes sure that cluster volumes are bounded and that a re-streaming is possible to improve the clustering. In the second phase of the algorithm, an edge partitioning is produced by exploiting the clustering to reduce the search space for possible block assignments to just two possible blocks. This makes the running time of the algorithm independent of k and therefore linear in the number of edges. First, the clusters are assigned to blocks such that each block has approximately the same total volume of clusters. The problem can be abstracted to the NP-hard makespan scheduling problem on identical machines [65, 66] and a $\frac{4}{3}$-approximation is received by running Graham's algorithm first [65]. Then a pre-partitioning is performed that assigns edges to blocks, where the block assignment of the edge endpoints' clusters does not differ. For the remaining edges, a scoring function is used that accounts for the two edge endpoints' degrees, the replication state, and the volume of the two endpoints' clusters to assign them to a block. A hard balance constraint makes sure no block is overloaded. By using this two-phase approach a more global view is obtained over the graph to build pre-partitions which increases the partitioning quality.

After having introduced the two competitors, we now compare our algorithm using the GBDTs model in a group generalization scenario that originates from Zwolak et al. [13]. We use the exact same graphs as published in GCNSplit's repository[4]. We train on the complete twitch-de graph and predict on the other twitch graphs provided and/or train on 10 000 edges of the deezer-ro graph and predict on the remaining deezer graphs just as in Zwolak et al. [13]. All graphs are power-law based graph instances from the same regime. We set the number of blocks to $k = 6$, the allowed imbalance to $\epsilon = 0.01$ and average the results for each graph over five runs. Note that for the results of GCNSplit, we are unfortunately forced to use the possibly slightly inexact reported results from Zwolak et al. [13], as we cannot set up the GCNSplit repository because of build script execution errors. The authors are aware of this issue but could not provide a working

[4]https://github.com/CASP-Systems-BU/GCNSplit

solution so far. The 2PS-L algorithm is run with its default configuration. Furthermore, the Fennel based baseline algorithm is also added to the comparison. We then obtain the following replication factor performances:

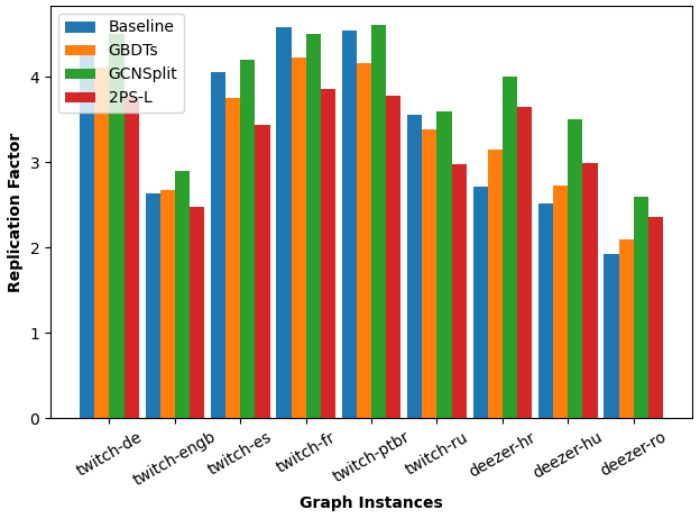


Figure 5.15: Competitor comparison: Replication factors for $k = 6$ and $\epsilon = 0.01$

We can see that using a heuristic feature based on Fennel scores combined with supervised learning and a GBDTs model, performs between 6.11% to 22.29% better than GCNSplit. We always outperform our machine learning based competitor by replication factor on the chosen graph instances. The comparison against 2PS-L is two-fold. On the twitch graphs, the 2PS-L algorithm gives between 7.46% to 11.83% lower replication factors, whereas our algorithm reports 9.03% to 13.80% lower replication factors on the bigger deezer graph instances. This might be because the bigger deezer graphs are more difficult to cluster for 2PS-L in a single streaming run. This experiment is also another proof that generalization to unseen graphs does work out, in case the generative model behind the graphs is exactly the same, here given by having provided graph instances originating from the same Twitch network dynamics. On the twitch graphs, our algorithm almost always outperforms the Fennel based baseline algorithm, whereas on the deezer graphs, our reported replication factors are 8.37% to 16.23% higher. This is possibly because 10 000 edges of the deezer-ro graph, which are less than 8% of its total number of edges, are not sufficient to learn the partitioning patterns of such graphs. We impose this restriction in order to be compliant with the experimental setup in GCNSplit's evaluation in Zwolak et al. [13].

CHAPTER 6

Discussion

6.1 Conclusion

In this thesis, we introduce a framework to utilize a variety of machine learning models from both conventional, explainable machine learning and deep learning origin as a heuristic for streaming graph partitioning. Our work presents, to the best of our knowledge, the first machine learning based streaming graph partitioning framework that is applicable to streams of any non-attributed graph as it constructs features purely from the structural properties of the graph. We are therefore no longer limited to partitioning solely attributed graphs or graphs from the same application domain. While the models are optimized for node streaming graph partitioning, our sliding-window based buffered streaming model also supports edge streaming. After tuning the framework and selecting the Fennel score based feature as the winner among all defined features within statistical, greedy, and heuristic feature groups, our predictions achieve generalization on evolving graph instances as well as on unseen graph instances that structurally match the training graph instances. The models' training and performance are based on supervised learning using block labels computed by the offline multilevel graph partitioner KaFFPa [14]. Furthermore, our novel concept of prediction propagation allows us to improve on previous block assignment predictions throughout the buffer in order to increase partitioning quality even further. In the end, our framework is able to outperform our main competitor GCNSplit by providing 6.11% to 22.29% lower replication factors in the given experiments. We recap all achievements and findings more in detail in the following.

After explaining the operating principle of all machine learning models available in our framework, we conduct rich literature research through the field of graph partitioning going from conventional algorithmic methodologies up to deep learning approaches for both conventional and streaming graph partitioning. Famous works include LDG [3], Fennel [4], AKIN [5] and HeiStream [9] from the conventional algorithmic streaming graph partitioning methods and GAP [10], its variant inspired by spectral partitioning [11], and GCNSplit [12, 13] from the deep learning approaches.

We introduce a sliding-window based buffered streaming model being able to adapt for variable buffer sizes and sliding window sizes. It also supports enriching nodes with additional features like cluster IDs from a label propagation algorithm. The model is constructed for both node and edge streams. The empirical evaluation shows a good quality / running time trade-off being achieved for buffer size $b = 256$, step size $s = 16$, and three prediction propagation rounds. During streaming along the natural order of nodes, features for each node in the buffer are constructed using only the graph itself and the current partitioning progress. Three feature groups are defined, i.e., statistical features, greedy features, and heuristic features, whereas the Fennel score based feature empirically turns out to perform best on average. Deciding for this feature also makes us combine elements from conventional streaming graph partitioning methods during feature creation with modern machine learning methods during prediction. As the feature is based on neighboring block assignments, the feature becomes more accurate the more the neighborhood is explored. We therefore add the concept of prediction propagation, meaning once a node gets assigned to a block, all neighboring nodes will update their feature vectors, their previous prediction will be checked again, and possibly improved. As the machine learning models cannot ensure balance, we evaluate two balance heuristics inspired by Abbas [12] and Zwolak et al. [13] that enforce a hard balance constraint and conclude the heuristic MostProbable to work best.

For the empirical evaluation studies, we construct two rich datasets, one for tuning and training and one for evaluation. Both of them are constructed to contain graphs from different groups of graphs having different structural properties and degree distributions. A degree distribution analysis reveals Gaussian based, regular, power-law based, and scale-free graphs. We then compare five different machine learning models and compare them to a Fennel based baseline algorithm. Among these models are the conventional machine learning models Logistic Regression, SVM and GBDTs and the deep learning

models GraphSAGE and Partitioner, a model inspired by Nazi et al. [10] and also used by Abbas [12] and Zwolak et al. [13]. In the rich feature selection experiments, we can confirm two hypotheses. First, the conventional machine learning models work best for features, that are directly indicative to which block to assign a node and so do the greedy and heuristic features outperform the statistical features on average. Second, deep learning models are capable to learn features on their own by extracting data intrinsic patterns and heuristics building upon these patterns. For such models, the statistical features notably become more important, even if not being able to outperform the greedy and heuristic feature group. In both model groups, the final selected feature is the Fennel score based feature because of its best performance among all other features on average, which should not exclude the applicability of other features for certain graph instances, e.g., those with a high variance in the degree distribution like email-EuAll. After tuning the architectures and hyperparameters, that do not show common patterns but are rather individual to each graph instance and its topology, we start the generalization experiments. It turns out that feature normalization and feature standardization especially helps the parametric models, which have an incorporated assumption about the data, to better learn their decision boundaries, whereas such preprocessing restricts the more sophisticated models to closely match the feature space.

While all models work decently well for instance generalization, group generalization is limited to such graph instances that are very similar in topology to those seen during the model training and is more successful the lower the value for the number of blocks k. In case the same generative model is used, generalization to unseen graphs even works out for bigger values of k. Generalizing to graphs of totally different origins is not feasible. For easy, evolving streaming graph partitioning settings with low values of k, one should preferably choose the more lightweight Logistic Regression classifier while for bigger k GBDTs achieve the best quality / running time trade-off besides GraphSAGE that gives comparable results but has longer training and prediction times. In the case of small datasets up to 10 000 nodes using an SVM instead of a Logistic Regression classifier also makes sense. In case the data to train on is larger, its quadratically scaling training effort nevertheless overweights its prediction benefits. The task of group generalization is always best addressed by the GBDTs using XGBoost's implementation [20]. In every case, using labeled data from KaFFPa [14] in a supervised learning setting is shown to have an influence on the error correction capability of the framework, blocks are less

clustered as compared to the Fennel based baseline algorithm and occurs more often the lower the number of blocks k.

In a final comparison to our competitors, we are able to consistently beat our machine learning based competitor GCNSplit [12, 13] with GBDTs giving between 6.11% to 22.29% lower replication factors. Compared to the algorithmic competitor 2PS-L [21], we obtain lower replication factors solely on the deezer graph instances, here the improvements range between 9.03% to 13.80%.

6.2 Future Work

As having set the requirement that the streaming scenario should act unknowingly of future nodes to be streamed, we always stream in the natural order of nodes as given by their node IDs. We know that streaming in BFS order can remarkably improve the partitioning quality and so future experiments shall explore more streaming orders, like ambivalence from Awadelkarim and Ugander [7]. In the case of a buffered streaming model, one might also consider the compromise to stick to the natural order of nodes during streaming, but execute ordering inside the buffer prior to prediction.

Also, all graphs used throughout this thesis are rather small graphs and enable quick feedback loops to further improve the algorithms and models. This allows us to stream the graphs from the main memory, that only partly covers the actual streaming graph partitioning application domain. Besides being used in online scenarios, partitioning huge graphs that do not fit into the main memory is the most common application area of streaming graph partitioning. Therefore, future work shall include experiments with huge graphs that are streamed from the disk. These experiments shall investigate the training efforts on the given hardware and the architectural model designs necessary to predict block assignments of such big graphs.

To speed up training efforts for such huge graphs, one might also investigate the field of online machine learning or incremental learning, where models are trained continuously once new data arrives. A discussion about the research opportunities in this area is given by Gomes et al. [67]. Besides this, there is also the field of out-of-core learning that one might consider investigating as it tries to learn on data that does not fit into the main memory.

Last but not least, to further speed up the framework for future experiments, an implementation in C++ is also imaginable. While especially the deep learning models are very hard to implement in C++ because of very limited graph based deep learning libraries available, XGBoost's implementation of GBDTs [20] is also supported for C++[1]. Here parallelization might also be implemented more easily such that a graph is processed by multiple processes at once that each holds a model for block assignment prediction. Scikit-learn's and PyTorch Geometric's training and prediction routines are already well parallelized, only the streaming and partitioning routine itself can be further improved through parallelization. Empirical evaluations of the throughput rate by the number of processes used would then be an interesting area to investigate further.

[1] https://xgboost.readthedocs.io/en/stable/

Bibliography

[1] Laurent Hyafil and Ronald L. Rivest. *Graph Partitioning and Constructing Optimal Decision Trees are Polynomial Complete Problems*. 1st ed. Vol. 33. IRIA - Laboratoire de Recherche en Informatique et Automatique, 1973. URL: https://people.csail.mit.edu/rivest/pubs/HR73.pdf (visited on 05/04/2022).

[2] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. "Some Simplified NP-Complete Graph Problems." In: *Theoretical Computer Science* 1.3 (1976), pp. 237–267. DOI: 10.1016/0304-3975(76)90059-1.

[3] Isabelle Stanton and Gabriel Kliot. "Streaming graph partitioning for large distributed graphs." In: *KDD '12: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Ed. by Qiang Yang, Deepak Agarwal, and Jian Pei. Beijing, China: ACM, 2012, pp. 1222–1230. DOI: 10.1145/2339530.2339722.

[4] Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnovic. "FENNEL: streaming graph partitioning for massive scale graphs." In: *WSDM '14: Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. Ed. by Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler. New York, NY, USA: ACM, 2014, pp. 333–342. DOI: 10.1145/2556195.2556213.

[5] Wei Zhang, Yong Chen, and Dong Dai. "AKIN: A Streaming Graph Partitioning Algorithm for Distributed Graph Storage Systems." In: *CCGrid '18: Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Ed. by Esam El-Araby, Dhabaleswar K. Panda, Sandra Gesing, Amy W. Apon, Volodymyr V. Kindratenko, Massimo Cafaro, and Alfredo Cuzzocrea.

Washington, DC, USA: IEEE Computer Society, 2018, pp. 183–192. DOI: 10.1109/CCGRID.2018.00033.

[6] Joel Nishimura and Johan Ugander. "Restreaming graph partitioning: simple versatile algorithms for advanced balancing." In: *KDD '13: The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Ed. by Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy. Chicago, IL, USA: ACM, 2013, pp. 1106–1114. DOI: 10.1145/2487575.2487696.

[7] Amel Awadelkarim and Johan Ugander. "Prioritized Restreaming Algorithms for Balanced Graph Partitioning." In: *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Ed. by Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash. Virtual Event, CA, USA: ACM, 2020, pp. 1877–1887. DOI: 10.1145/3394486.3403239.

[8] Md Anwarul Kaium Patwary, Saurabh Kumar Garg, and Byeong Kang. "Window-based Streaming Graph Partitioning Algorithm." In: *ACSW '19: Proceedings of the Australasian Computer Science Week Multiconference*. Sydney, NSW, Australia: ACM, 2019, 51:1–51:10. DOI: 10.1145/3290688.3290711.

[9] Marcelo Fonseca Faraj and Christian Schulz. "Buffered Streaming Graph Partitioning." In: *ACM Journal of Experimental Algorithmics* (2022). DOI: 10.1145/3546911.

[10] Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. "GAP: Generalizable Approximate Graph Partitioning Framework." In: *CoRR* abs/1903.00614 (2019). DOI: 10.48550/arXiv.1903.00614.

[11] Alice Gatti, Zhixiong Hu, Tess E. Smidt, Esmond G. Ng, and Pieter Ghysels. "Deep Learning and Spectral Embedding for Graph Partitioning." In: *PPSC '22: Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing*. Ed. by Xiaoye S. Li and Keita Teranishi. Seattle, WA, USA, SIAM, 2022, pp. 25–36. DOI: 10.1137/1.9781611977141.3.

[12] Zainab Abbas. "Scalable Streaming Graph and Time Series Analysis Using Partitioning and Machine Learning." PhD thesis. Stockholm, Sweden: Royal Institute of Technology, 2021. URL: https://nbn-resolving.org/urn:nbn:se:kth:diva-301596 (visited on 03/18/2022).

[13] Michal Zwolak, Zainab Abbas, Sonia Horchidan, Paris Carbone, and Vasiliki Kalavri. "GCNSplit: bounding the state of streaming graph partitioning." In: *aiDM '22: Proceedings of the Fifth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. Ed. by Rajesh Bordawekar, Oded Shmueli, Yael Amsterdamer, Donatella Firmani, and Ryan Marcus. Philadelphia, Pennsylvania, USA: ACM, 2022, 3:1–3:12. DOI: 10.1145/3533702.3534920.

[14] Peter Sanders and Christian Schulz. "Think Locally, Act Globally: Highly Balanced Graph Partitioning." In: *SEA '13: Proceedings of the 12th International Symposium on Experimental Algorithms*. Ed. by Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela. Vol. 7933. Lecture Notes in Computer Science. Rome, Italy: Springer Berlin Heidelberg, 2013, pp. 164–175. DOI: 10.1007/978-3-642-38527-8_16.

[15] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. "Recent Advances in Graph Partitioning." In: *Algorithm Engineering - Selected Results and Surveys*. Ed. by Lasse Kliemann and Peter Sanders. Vol. 9220. Lecture Notes in Computer Science. Springer, 2016, pp. 117–158. DOI: 10.1007/978-3-319-49487-6_4.

[16] Charles J. Alpert and Andrew B. Kahng. "Recent directions in netlist partitioning: a survey." In: *Integration* 19.1 (1995), pp. 1–81. DOI: 10.1016/0167-9260(95)00008-4.

[17] Ümit V. Çatalyürek, Karen D. Devine, Marcelo Fonseca Faraj, Lars Gottesbüren, Tobias Heuer, Henning Meyerhenke, Peter Sanders, Sebastian Schlag, Christian Schulz, Daniel Seemaier, and Dorothea Wagner. "More Recent Advances in (Hyper)Graph Partitioning." In: *CoRR* abs/2205.13202 (2022). DOI: 10.48550/arXiv.2205.13202.

[18] Thang Nguyen Bui and Curt Jones. "Finding Good Approximate Vertex and Edge Partitions is NP-Hard." In: *Information Processing Letters* 42.3 (1992), pp. 153–159. DOI: 10.1016/0020-0190(92)90140-Q.

[19] Nazanin Jafari, Oguz Selvitopi, and Cevdet Aykanat. "Fast shared-memory streaming multilevel graph partitioning." In: *Journal of Parallel and Distributed Computing* 147 (2021), pp. 140–151. DOI: 10.1016/j.jpdc.2020.09.004.

[20] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Ed. by Balaji Krishnapuram, Mohak Shah,

Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi. San Francisco, CA, USA: ACM, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[21] Ruben Mayer, Kamil Orujzade, and Hans-Arno Jacobsen. "Out-of-Core Edge Partitioning at Linear Run-Time." In: *ICDE '22: Proceedings of the 38th IEEE International Conference on Data Engineering*. Kuala Lumpur, Malaysia: IEEE, 2022, pp. 2629–2642. DOI: 10.1109/ICDE53745.2022.00242.

[22] Aurélien Géron. *Machine Learning mit Scikit-Learn & Tensorflow*. 1st ed. O'Reilly, 2017. URL: https://www.oreilly.com/library/view/praxiseinstieg-machine-learning/9781492065838/ (visited on 07/04/2022).

[23] Terence Parr and Jeremy Howard. *How to explain gradient boosting*. explained.ai. 2022. URL: https://explained.ai/gradient-boosting/ (visited on 07/06/2022).

[24] Vihar Kurama. *Gradient Boosting In Classification: Not a Black Box Anymore!* Paperspace Blog. 2020. URL: https://blog.paperspace.com/gradient-boosting-for-classification/ (visited on 07/06/2022).

[25] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." In: *NIPS '17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. Long Beach, CA, USA: ACM, 2017, pp. 3146–3154. DOI: 10.5555/3294996.3295074.

[26] Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. "CatBoost: unbiased boosting with categorical features." In: *NIPS '18: Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. Montréal, Canada: ACM, 2018, pp. 6639–6649. DOI: 10.5555/3327757.3327770.

[27] Jair Cervantes, Farid García-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrúbal López Chau. "A comprehensive survey on support vector machine classification: Applications, challenges and trends." In: *Neurocomputing* 408 (SI: Advanced Intelligent Computing Theory and Applications 2020), pp. 189–215. DOI: 10.1016/j.neucom.2019.10.118.

[28] Andrew Zisserman. "SVM dual, kernels and regression." Oxford University, 2015. URL: https://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf (visited on 05/09/2022).

[29] Sebastian Schlag, Matthias Schmitt, and Christian Schulz. "Faster Support Vector Machines." In: *ACM Journal of Experimental Algorithmics* 26.15 (2021), pp. 1–21. DOI: 10.1145/3484730.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: online learning of social representations." In: *KDD '14: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Ed. by Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani. New York, NY, USA: ACM, 2014, pp. 701–710. DOI: 10.1145/2623330.2623732.

[31] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks." In: *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Ed. by Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi. San Francisco, CA, USA: ACM, 2016, pp. 855–864. DOI: 10.1145/2939672.2939754.

[32] William L. Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs." In: *NIPS '17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. Long Beach, CA, USA: ACM, 2017, pp. 1024–1034. DOI: 10.5555/3294771.3294869.

[33] Bruce Hendrickson and Robert Leland. "A Multilevel Algorithm for Partitioning Graphs." In: *Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. Supercomputing '95. San Diego, CA, USA: ACM, 1995, 28–es. DOI: 10.1145/224170.224228.

[34] Jin Kim, Inwook Hwang, Yong-Hyuk Kim, and Byung Ro Moon. "Genetic approaches for graph partitioning: a survey." In: *GECCO '11: Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference*. Ed. by Natalio Krasnogor and Pier Luca Lanzi. Dublin, Ireland: ACM, 2011, pp. 473–480. DOI: 10.1145/2001576.2001642.

[35] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed. Addison-Wesley Longman Publishing Co., Inc., 1989. URL: https://dl.acm.org/doi/10.5555/534133 (visited on 09/07/2022).

[36] Charles-Edmond Bichot and Patrick Siarry. *Graph Partitioning*. 1st ed. ISTE. Wiley, 2013. DOI: 10.1002/9781118601181.

[37] Zainab Abbas, Vasiliki Kalavri, Paris Carbone, and Vladimir Vlassov. "Streaming Graph Partitioning: An Experimental Study." In: *Proceedings of the VLDB Endowment* 11.11 (2018), pp. 1590–1603. DOI: 10.14778/3236187.3236208.

[38] Anil Pacaci and M. Tamer Özsu. "Experimental Analysis of Streaming Algorithms for Graph Partitioning." In: *SIGMOD '19: Proceedings of the 2019 International Conference on Management of Data*. Ed. by Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska. Amsterdam, The Netherlands: ACM, 2019, pp. 1375–1392. DOI: 10.1145/3299869.3300076.

[39] Lieve Hamers, Yves Hemeryck, Guido Herweyers, Marc Janssen, Hans Keters, Ronald Rousseau, and André Vanhoutte. "Similarity measures in scientometric research: The Jaccard index versus Salton's cosine formula." In: *Information Processing & Management* 25.3 (1989), pp. 315–318. DOI: 10.1016/0306-4573(89)90048-4.

[40] Reza Khamoushi. *Scalable Streaming Graph Partitioning*. Master's Thesis. Royal Institute of Technology, Stockholm, Sweden, 2017. URL: https://nbn-resolving.org/urn:nbn:se:kth:diva-206113 (visited on 03/30/2022).

[41] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In: *ICLR '17: Proceedings of the 5th International Conference on Learning Representations*. Toulon, France: OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl (visited on 05/31/2022).

[42] George Karypis and Vipin Kumar. "Multilevel k-way Hypergraph Partitioning." In: *VLSI Design* 2000.3 (2000), pp. 285–300. DOI: 10.1155/2000/19436.

[43] William L. Briggs, Van Emden Henson, and Stephen F. McCormick. *A multigrid tutorial, Second Edition*. 1st ed. SIAM, 2000. DOI: 10.1137/1.9780898719505.

[44] Miroslav Fiedler. "Algebraic connectivity of graphs." In: *Czechoslovak Mathematical Journal* 23.2 (1971), pp. 298–305. DOI: 10.21136/CMJ.1973.101168.

[45] Matthias Fey and Jan Eric Lenssen. "Fast Graph Representation Learning with PyTorch Geometric." In: *CoRR* abs/1903.02428 (2019). DOI: 10.48550/arXiv.1903.02428.

[46] George Karypis and Vipin Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs." In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392. DOI: 10.1137/S1064827595287997.

[47] François Pellegrini and Jean Roman. "SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs." In: *HPCN '96: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*. Ed. by Heather M. Liddell, Adrian Colbrook, Louis O. Hertzberger, and Peter M. A. Sloot. Vol. 1067. Lecture Notes in Computer Science. Brussels, Belgium: Springer, 1996, pp. 493–498. DOI: 10.1007/3-540-61142-8_588.

[48] Elizabeth D. Dolan and Jorge J. Moré. "Benchmarking optimization software with performance profiles." In: *Mathematical Programming* 91.2 (2002), pp. 201–213. DOI: 10.1007/s101070100263.

[49] Michal Zwolak. *Streaming Graph Partitioning with Graph Convolutional Networks*. Master's Thesis. Royal Institute of Technology, Stockholm, Sweden, 2020. URL: https://nbn-resolving.org/urn:nbn:se:kth:diva-280320 (visited on 03/18/2022).

[50] Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. "HDRF: Stream-Based Partitioning for Power-Law Graphs." In: *CIKM '15: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. Ed. by James Bailey, Alistair Moffat, Charu C. Aggarwal, Maarten de Rijke, Ravi Kumar, Vanessa Murdock, Timos K. Sellis, and Jeffrey Xu Yu. Melbourne, VIC, Australia: ACM, 2015, pp. 243–252. DOI: 10.1145/2806416.2806424.

[51] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks." In: *Physical Review E* 76.3 (2007), p. 036106. DOI: 10.1103/PhysRevE.76.036106.

[52] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu

Brucher, Matthieu Perrot, and Edouard Duchesnay. "Scikit-learn: Machine Learning in Python." In: *The Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *NIPS '19: Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Vancouver, BC, Canada. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett. ACM, 2019, pp. 8024–8035. DOI: 10.5555/3454287.3455008.

[54] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *ICLR '15: Proceedings of the 3rd International Conference on Learning Representations*. Ed. by Yoshua Bengio and Yann LeCun. San Diego, CA, USA: OpenReview.net, 2015. URL: https://openreview.net/forum?id=8gmWwjFyLj (visited on 08/17/2022).

[55] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *AISTATS '10: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and D. Mike Titterington. Vol. 9. JMLR Proceedings. Chia Laguna Resort, Sardinia, Italy: JMLR.org, 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html (visited on 06/18/2022).

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *ICCV '15: Proceedings of the 2015 IEEE International Conference on Computer Vision*. Santiago, Chile: IEEE Computer Society, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[57] David A. Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. "Benchmarking for Graph Clustering and Partitioning." In: *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. Ed. by Reda Alhajj and Jon G. Rokne. Springer, 2018, pp. 161–171. DOI: 10.1007/978-1-4939-7131-2_23.

[58] Alan J. Soper, Chris Walshaw, and Mark Cross. "A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning." In: *Journal of Global Optimization* 29.2 (2004), pp. 225–241. DOI: 10.1023/B:JOGO.0000042115.44455.f3.

[59] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization." In: *AAAI '15: Proceedings of the 29th AAAI Conference on Artificial Intelligence*. Ed. by Blai Bonet and Sven Koenig. Austin, Texas, USA: ACM, 2015, pp. 4292–4293. DOI: 10.5555/2888116.2888372.

[60] Jure Leskovec, Andrej Krevl, and Rok Sosic. "SNAP Datasets: Stanford Large Network Dataset Collection." In: *CoRR* abs/1606.07550 (2016). DOI: 10.48550/arXiv.1606.07550.

[61] Paolo Boldi and Sebastiano Vigna. "The webgraph framework I: compression techniques." In: *WWW '04: Proceedings of the 13th International Conference on World Wide Web*. Ed. by Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills. New York, NY, USA, ACM, 2004, pp. 595–602. DOI: 10.1145/988672.988752.

[62] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE." In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html (visited on 08/18/2022).

[63] Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. "Communication-free massively distributed graph generation." In: *Journal of Parallel and Distributed Computing* 131 (SI: Selected papers from IPDPS'18 2019), pp. 200–217. DOI: 10.1016/j.jpdc.2019.03.011.

[64] Alexandre Hollocou, Julien Maudet, Thomas Bonald, and Marc Lelarge. "A Streaming Algorithm for Graph Clustering." In: *CoRR* abs/1712.04337 (2017). DOI: 10.48550/arXiv.1712.04337.

[65] Ronald L. Graham. "Bounds on Multiprocessing Timing Anomalies." In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429. DOI: 10.1137/0117039.

[66] Jeffrey D. Ullman. "NP-Complete Scheduling Problems." In: *Journal of Computer and System Sciences* 10.3 (1975), pp. 384–393. DOI: 10.1016/S0022-0000(75)80008-0.

[67] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. "Machine learning for streaming data: state of the art, challenges, and opportunities." In: *ACM SIGKDD Explorations Newsletter* 21.2 (2019), pp. 6–22. DOI: 10.1145/3373464.3373470.

APPENDIX A

Datasets

In this chapter, we further introduce our two selected datasets for the tuning and evaluation phase of the experimental evaluation and especially focus on a degree distribution analysis. The experimental dataset is presented in Table A.1 and contains graphs in the order of around 10 000 nodes.

Group	**Graph**	**Nodes**	**Edges**
artificial	delaunay_n13	8 192	24 547
artificial	delaunay_n14	16 384	49 122
coauthorship	astro-ph	16 706	121 251
coauthorship	cond-mat	16 726	47 594
meshes	fe_4elt2	11 143	32 818
meshes	fe_sphere	16 386	49 152
road	road-euroroad	1 174	1 417
road	road-minnesota	2 642	3 303
social	email-EuAll	16 805	60 260
social	wordassociation-2011	10 617	63 788

Table A.1: Tuning Set

The degree distributions of the graphs inside the tuning dataset are shown below. We recognize a Gaussian-like distribution on the generated artificial graphs (see Figure A.1) with a mean of six and a standard deviation of 1.34. The mesh-type graphs (see Figure A.3)

tend to have a more regular structure, more than 60% of the nodes from fe_4elt2 have a degree of six and the standard deviation is less than one. Fe_sphere even has solely nodes of degree six and is therefore a true regular graph. The technical road networks (see Figure A.4) have a structural limitation by nature, in both of them more than 50% of the nodes have a degree of two, road-minnesota does not surpass a maximum degree of five, and the percentage of nodes having a degree higher than five becomes also negligible for road-euroroad. A power-law degree distribution can be fitted for the coauthorship graphs (see Figure A.2) as well as for the social graphs (see Figure A.5).

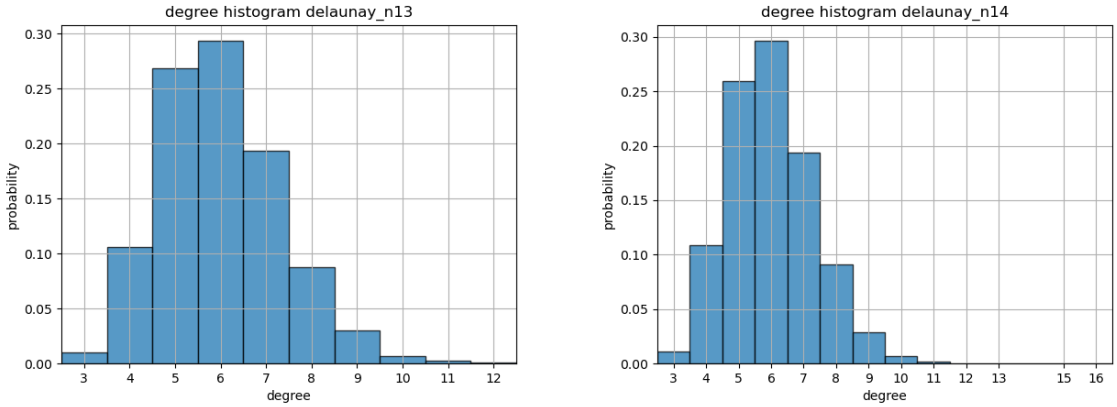


Figure A.1: Degree distributions of the tuning dataset: Artificial graphs

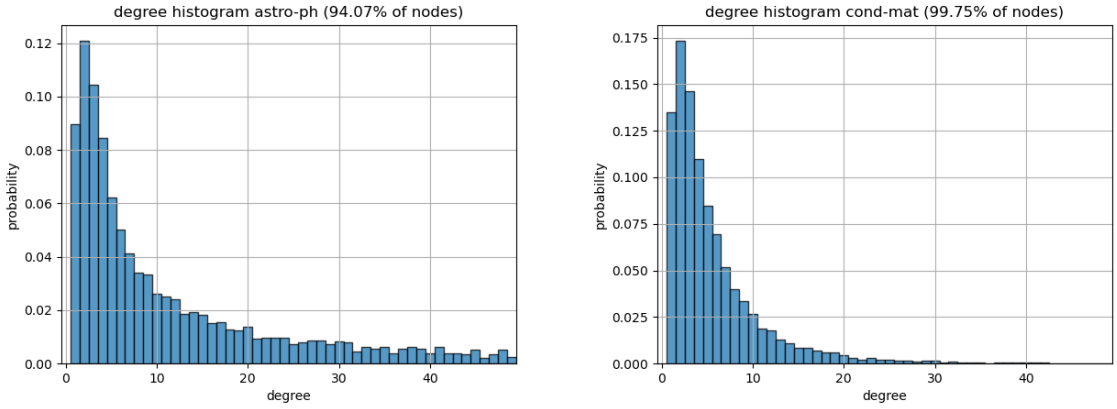


Figure A.2: Degree distributions of the tuning dataset: Coauthorship graphs

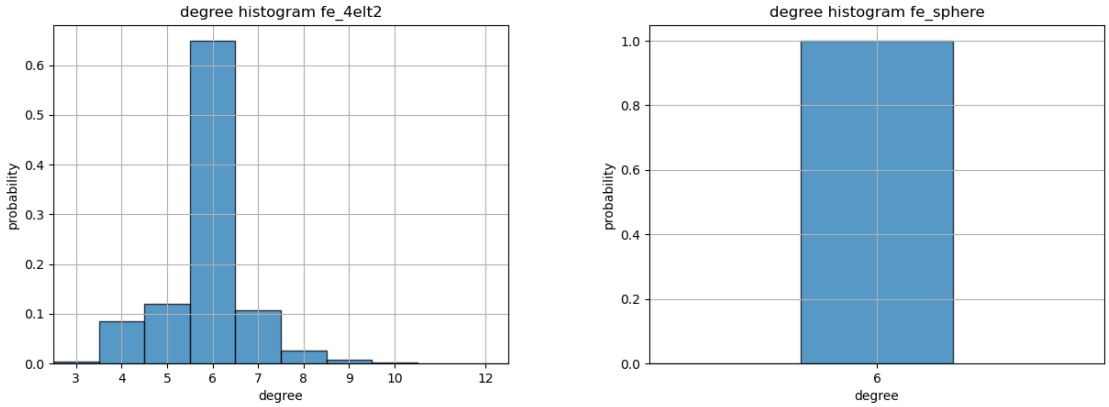


Figure A.3: Degree distributions of the tuning dataset: Mesh graphs

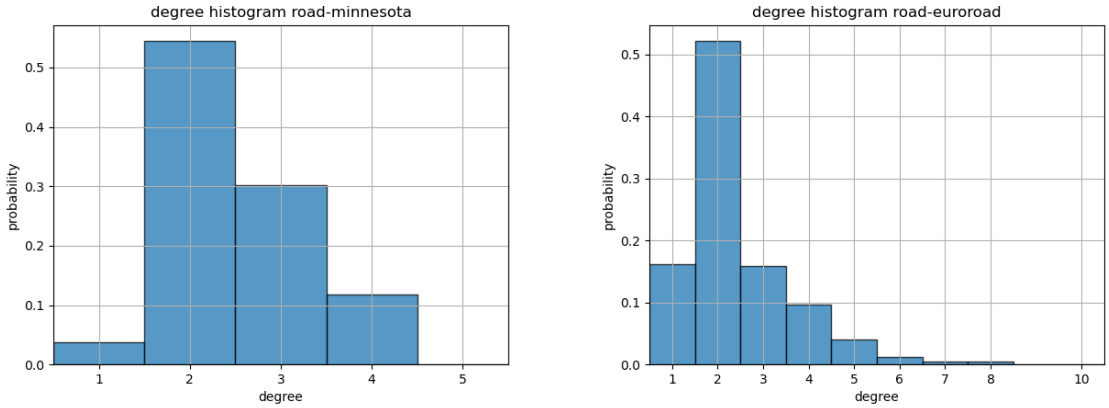


Figure A.4: Degree distributions of the tuning dataset: Road graphs

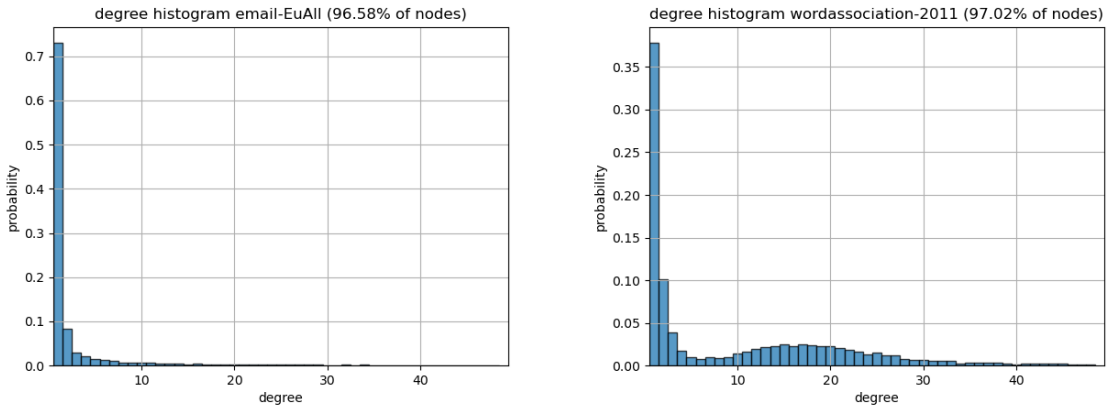


Figure A.5: Degree distributions of the tuning dataset: Social graphs

A true scale-free graph classification can only be determined for email-EuAll with an approximate degree exponent of $\gamma = 2.64$. Also one can conclude a few hubs from the tail of the degree distribution in Figure A.6. The remaining coauthorship graphs and social graphs are all located in the anomalous regime with a degree exponent γ smaller than two.

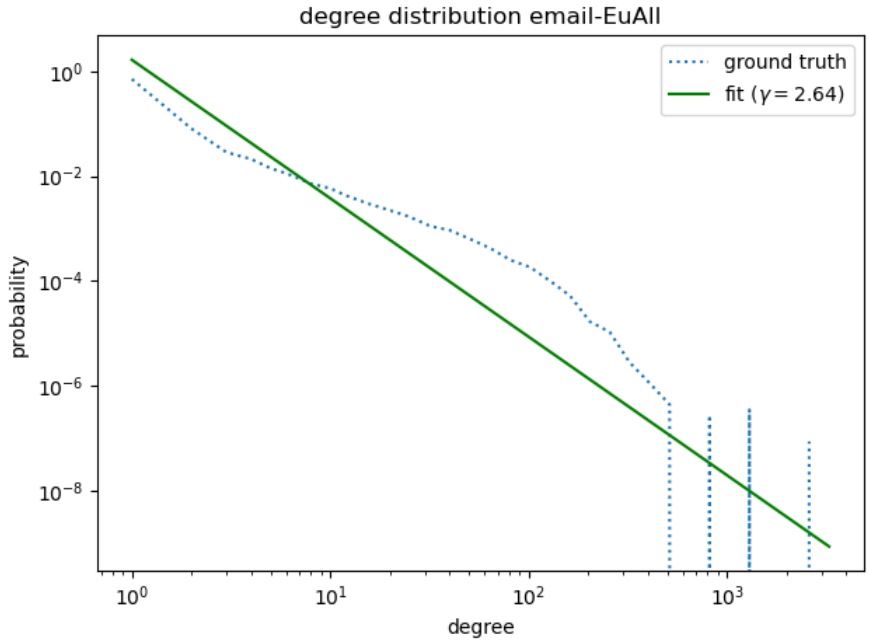


Figure A.6: Fitted scale-free power law distribution for email-EuAll

The dataset used to evaluate the tuned framework from the tuning graphs is given in Table A.2 and contains slightly larger instances between approximately 20 000 and 40 000 nodes. During the design, it is important to guarantee that these graphs can be assigned to the same structural and domain-specific groups as seen for the tuning dataset.

Group	Graph	Nodes	Edges
artificial	delaunay_n15	32 768	98 274
artificial	rgg_n_2_15_s0	32 768	160 240
coauthorship	cond-mat-2003	31 163	120 029
coauthorship	cond-mat-2005	40 421	175 691
meshes	bcsstk30	28 924	1 007 284
meshes	cs4	22 499	43 858
social	as-22july06	22 963	48 436
social	soc-Slashdot0902	28 550	379 445

Table A.2: Evaluation Set

Again the degree distributions of the different graphs are shown below. Note that we cannot find road graphs in the same size dimension. Anyways the previous degree distributions of the road graphs do not differ a lot from, e.g., the artificial graphs, such that they can mimic their structural properties.

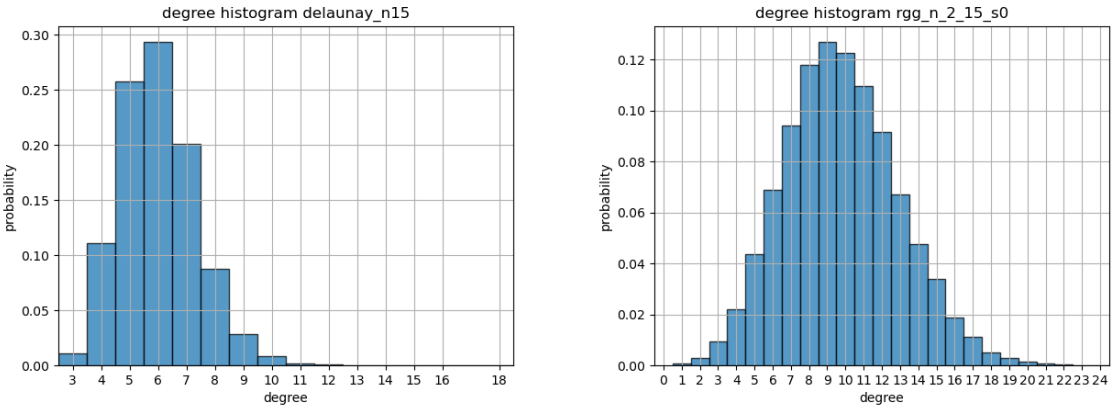


Figure A.7: Degree distributions of the evaluation dataset: Artificial graphs

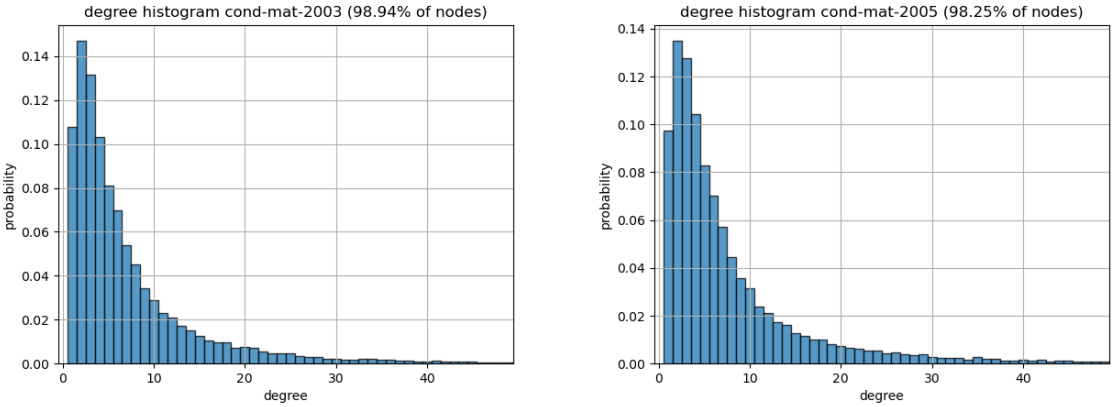


Figure A.8: Degree distributions of the evaluation dataset: Coauthorship graphs

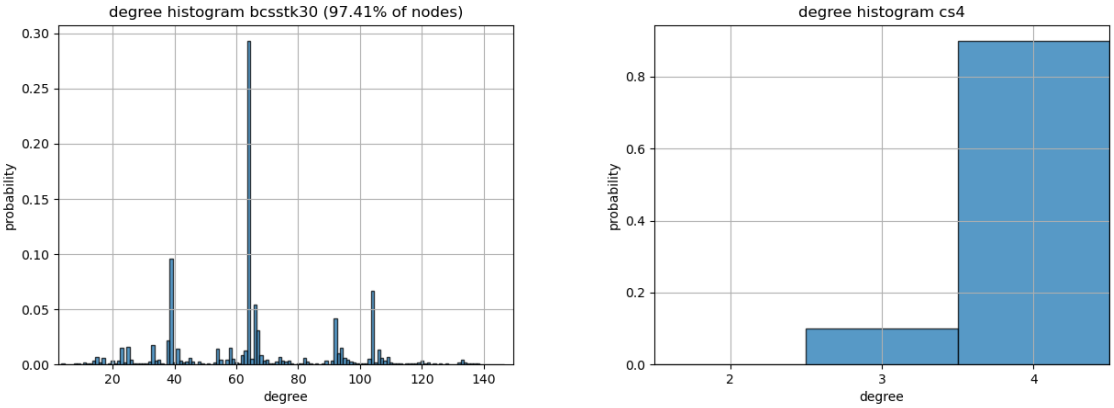


Figure A.9: Degree distributions of the evaluation dataset: Mesh graphs

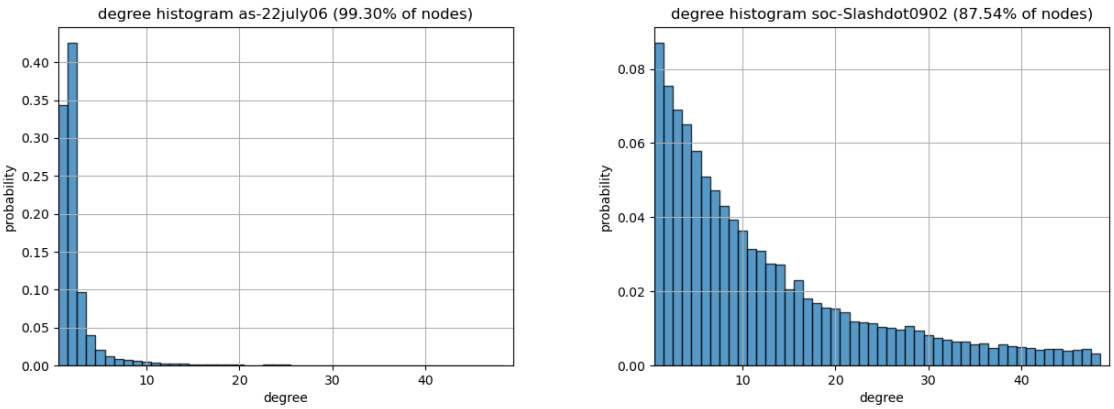


Figure A.10: Degree distributions of the evaluation dataset: Social graphs

Once more the artificial graphs (see Figure A.7) follow a Gaussian distribution with the delaunay_n15 graph having a mean of six and a standard deviation of 1.36 and the rgg_n_2_15_s0 graph with a mean of 9.78 and a standard deviation of 3.16. From the mesh-type graphs (see Figure A.9), only the cs4 graph seems to have a regular tendency with more than 84% of the nodes having a degree of four. The bcsstk30 graph on the other hand, which represent a stiffness matrix, is completely unregular and distributes the node degrees seemingly randomly over an interval reaching up to a maximum degree of 218. A power-law based degree distribution can again be fitted for the coauthorship (see Figure A.8) and social graphs (see Figure A.10).

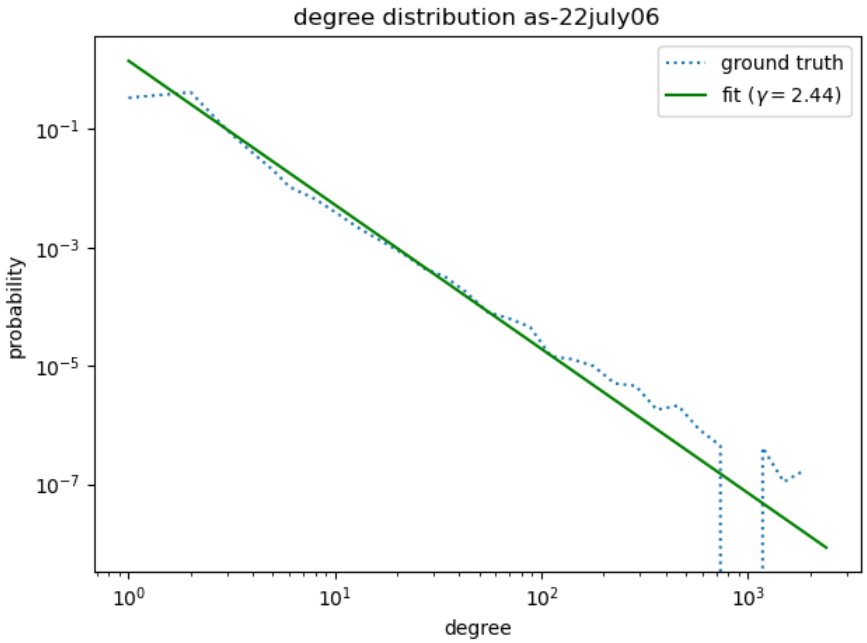


Figure A.11: Fitted scale-free power law distribution for as-22july06

Figure A.11 shows the fitted power-law distribution for the as-22july06 graph, which is the only one in the evaluation dataset being scale-free with an approximate degree exponent of $\gamma = 2.44$. The remaining power-law based graphs are again from the anomalous regime.

APPENDIX B

Feature Selection

Graph	KaFFPa Eco	Baseline	Statistical	Greedy	Heuristic
delaunay_n13	1 045	4 381	4 387	**4 007.8**	4 077.8
delaunay_n14	1 462	7 989	7 342	**7 202.6**	7 792.4
astro-ph	26 066	**39 642**	40 935	40 352	39 987.4
cond-mat	5 663	**11 348**	11 886	11 934.6	11 548.2
fe_4elt2	1 127	**6 115**	7 246.4	6 693	6 373.8
fe_sphere	1 959	3 885	5 857.6	3 763	**3 461.8**
road-minnesota	141	387	507.4	369	**361.8**
road-euroroad	100	260	299	**253**	256.8
email-EuAll	18 378	27 662	**25 811**	28 495	27 467.8
wordassociation-2011	31 266	36 168	38 103	**35 914.6**	36 061.8
Geometric Mean		5 722.45	6 304.84	5 655.11	**5 590.07**

Table B.1: Logistic Regression feature evaluation: Total cut sizes by feature group

Graph	Baseline	NC+PH	NC	PH
delaunay_n13	**4 381**	4 387	4 663.4 (+)	4 500 (+)
delaunay_n14	7 989	**7 342**	8 529.4 (+)	7 859 (+)
astro-ph	39 642	40 935	40 743 (-)	**39 044 (-)**
cond-mat	**11 348**	11 886	12 166.4 (+)	11 786 (-)
fe_4elt2	**6 115**	7 246.4	7 160.6 (-)	7 470 (+)
fe_sphere	**3 885**	5 857.6	4 549 (-)	6 086.6 (+)
road-minnesota	**387**	507.4	561.8 (+)	481 (-)
road-euroroad	**260**	299	289.2 (-)	296 (-)
email-EuAll	27 662	**25 811**	25 885 (+)	28 188.2 (+)
wordassociation-2011	**36 168**	38 103	37 454.4 (-)	38 155 (+)
Geometric Mean	**5 722.45**	6 304.84	6 316.89	6 388.88

Table B.2: Logistic Regression feature evaluation: Total cut sizes by statistical feature

Graph	**Baseline**	**NBC+EC+NB**	**EC+NB**
delaunay_n13	4 381	4 007.8	3 972.8 (-)
delaunay_n14	7 989	**7 202.6**	7 243.8 (+)
astro-ph	**39 642**	40 352	40 330 (-)
cond-mat	**11 348**	11 934.6	11 964.4 (+)
fe_4elt2	**6 115**	6 693	6 705.4 (+)
fe_sphere	3 885	3 763	3 817.2 (+)
road-minnesota	387	369	370 (+)
road-euroroad	260	253	**250 (-)**
email-EuAll	**27 662**	28 495	28 505 (+)
wordassociation-2011	36 168	35 914.6	**35 617.4 (-)**
Geometric Mean	5 722.45	5 655.11	5 659.67
Graph	**NBC**	**EC**	**NB**
delaunay_n13	4 106.8 (+)	**3 880 (-)**	4 095 (+)
delaunay_n14	7 525.2 (+)	7 256.4 (+)	7 313.2 (+)
astro-ph	40 561.2 (+)	40 127.8 (-)	40 519.2 (+)
cond-mat	11 875.4 (-)	11 856.8 (-)	11 840 (-)
fe_4elt2	6 963.6 (+)	6 819.4 (+)	6 851.2 (+)
fe_sphere	3 976 (+)	**3 542.6 (-)**	4 009.2 (+)
road-minnesota	394.4 (+)	**364.8 (-)**	367.4 (-)
road-euroroad	255 (+)	253.6 (+)	256.6 (+)
email-EuAll	27 772.4 (-)	28 443 (-)	27 756 (-)
wordassociation-2011	36 050.2 (+)	35 865.2 (-)	36 142 (+)
Geometric Mean	5 778.38	**5 603.88**	5 717.37

Table B.3: Logistic Regression feature evaluation: Total cut sizes by greedy feature

Graph	Baseline	Fennel+LDG	Fennel	LDG
delaunay_n13	4 381	**4 077.8**	4 209 (+)	4 173.4 (+)
delaunay_n14	7 989	7 792.4	7 735.6 (-)	**7 386 (-)**
astro-ph	**39 642**	39 987.4	39 855 (-)	40 525 (+)
cond-mat	**11 348**	11 548.2	11 569.8 (+)	11 926 (+)
fe_4elt2	**6 115**	6 373.8	6 018.8 (-)	6 829 (+)
fe_sphere	3 885	3 461.8	**3 343.4 (-)**	4 203.8 (+)
road-minnesota	387	361.8	**361.6 (-)**	372.4 (+)
road-euroroad	260	256.8	**254.4 (-)**	256.6 (-)
email-EuAll	27 662	27 467.8	**26 668 (-)**	28 362 (+)
wordassociation-2011	36 168	**36 061.8**	36 102.4 (+)	36 539.8 (+)
Geometric Mean	5 722.45	5 590.07	**5 530.26**	5 790.14

Table B.4: Logistic Regression feature evaluation: Total cut sizes by heuristic feature

Graph	Baseline	NC+PH	EC	Fennel	EC+Fennel
delaunay_n13	4 381	4 387	**3 880**	4 209	4 184
delaunay_n14	7 989	7 342	**7 256.4**	7 735.6	7 679
astro-ph	**39 642**	40 935	40 127.8	39 855	39 433.8
cond-mat	**11 348**	11 886	11 856.8	11 569.8	11 511.8
fe_4elt2	6 115	7 246.4	6 819.4	**6 018.8**	6 435.2
fe_sphere	3 885	5 857.6	3 542.6	**3 343.4**	3 532.8
road-minnesota	387	507.4	364.8	361.6	**361.2**
road-euroroad	260	299	**253.6**	254.4	254.8
email-EuAll	27 662	25 811	28 443	26 668	**24 568.2**
wordassociation-2011	36 168	38 103	**35 865.2**	36 102.4	36 023.6
Geometric Mean	5 722.45	6 304.84	5 603.88	**5 530.26**	5 535.41

Table B.5: Logistic Regression feature evaluation: Total cut sizes by selected features

Graph	KaFFPa Eco	Baseline	Statistical	Greedy	Heuristic
delaunay_n13	1 045	4 381	4 521	4 232.4	**4 141**
delaunay_n14	1 462	7 989	**7 110**	7 358	7 746
astro-ph	26 066	**39 642**	40 599	40 854.2	40 763
cond-mat	5 663	**11 348**	11 943	11 848.4	11 693
fe_4elt2	1 127	**6 115**	7 106	6 659.8	6 250
fe_sphere	1 959	3 885	6 151	3 891.6	**3 493**
road-minnesota	141	387	480	375.8	**347**
road-euroroad	100	260	287.6	**256.4**	257.4
email-EuAll	18 378	27 662	**26 000**	26 963.6	27 739
wordassociation-2011	31 266	36 168	37 513	**36 107.6**	36 353
Geometric Mean		5 722.45	6 255.31	5 707.01	**5 595.03**

Table B.6: GBDTs feature evaluation: Total cut sizes by feature group

Graph	Baseline	NC+PH	NC	PH
delaunay_n13	**4 381**	4 521	4 504 (-)	4 448 (-)
delaunay_n14	7 989	**7 110**	7 735 (+)	7 660 (+)
astro-ph	**39 642**	40 599	40 731 (+)	40 646 (+)
cond-mat	**11 348**	11 943	12 084 (+)	11 740 (-)
fe_4elt2	**6 115**	7 106	7 953 (+)	7 533 (+)
fe_sphere	**3 885**	6 151	4 645 (-)	5 926 (-)
road-minnesota	**387**	480	533 (+)	476 (-)
road-euroroad	**260**	287.6	290.2 (+)	299.2 (+)
email-EuAll	27 662	**26 000**	26 788 (+)	28 500 (+)
wordassociation-2011	**36 168**	37 513	36 823 (-)	37 735 (+)
Geometric Mean	**5 722.45**	6 255.31	6 288.10	6 376.71

Table B.7: GBDTs feature evaluation: Total cut sizes by statistical feature

Graph	**Baseline**	**NBC+EC+NB**	**EC+NB**
delaunay_n13	4 381	4 232.4	4 186 (-)
delaunay_n14	7 989	7 358	6 981 (-)
astro-ph	**39 642**	40 854.2	40 908 (+)
cond-mat	**11 348**	11 848.4	11 875 (+)
fe_4elt2	**6 115**	6 659.8	6 592 (-)
fe_sphere	**3 885**	3 891.6	3 870 (-)
road-minnesota	387	**375.8**	376 (+)
road-euroroad	260	256.4	255.6 (-)
email-EuAll	27 662	26 963.6	26 725 (-)
wordassociation-2011	36 168	36 107.6	**36 018 (-)**
Geometric Mean	5 722.45	5 707.01	**5 655.97**
Graph	**NBC**	**EC**	**NB**
delaunay_n13	4 323.8 (+)	**4 169 (-)**	4 309 (+)
delaunay_n14	7 505.8 (+)	7 406 (+)	**6 756 (-)**
astro-ph	40 483.8 (-)	40 804 (-)	39 920 (-)
cond-mat	11 944.8 (+)	11 897 (+)	11 890 (+)
fe_4elt2	6 972.2 (+)	6 790 (+)	7 061 (+)
fe_sphere	3 964 (+)	**3 848 (-)**	3 970 (+)
road-minnesota	393.8 (+)	390 (+)	385 (+)
road-euroroad	259 (+)	255.8 (-)	**254.2 (-)**
email-EuAll	26 782.8 (-)	28 068 (+)	**26 663 (-)**
wordassociation-2011	36 167.8 (+)	36 368 (+)	36 177 (+)
Geometric Mean	5 796.93	5 755.40	5 705.68

Table B.8: GBDTs feature evaluation: Total cut sizes by greedy feature

Graph	Baseline	Fennel+LDG	Fennel	LDG
delaunay_n13	4 381	4 141	**4 063 (-)**	4 284 (+)
delaunay_n14	7 989	7 746	7 264 (-)	**7 175 (-)**
astro-ph	**39 642**	40 763	40 020 (-)	40 683 (-)
cond-mat	**11 348**	11 693	11 548 (-)	11 867 (+)
fe_4elt2	**6 115**	6 250	6 370 (+)	6 860 (+)
fe_sphere	3 885	3 493	**3 342 (-)**	4 631 (+)
road-minnesota	387	**347**	373 (+)	372 (+)
road-euroroad	260	**257.4**	262.2 (+)	264.8 (+)
email-EuAll	27 662	27 739	**26 178 (-)**	27 776 (+)
wordassociation-2011	36 168	36 353	**36 084 (-)**	36 944 (+)
Geometric Mean	5 722.45	5 595.03	**5 531.65**	5 858.82

Table B.9: GBDTs feature evaluation: Total cut sizes by heuristic feature

Graph	Baseline	NC+PH	EC+NB	Fennel	EC+NB+Fennel
delaunay_n13	4 381	4 521	4 186	4 063	**4 050**
delaunay_n14	7 989	7 110	**6 981**	7 264	7 459
astro-ph	**39 642**	40 599	40 908	40 020	40 722
cond-mat	**11 348**	11 943	11 875	11 548	11 849
fe_4elt2	**6 115**	7 106	6 592	6 370	6 473
fe_sphere	3 885	6 151	3 870	**3 342**	3 688
road-minnesota	387	480	376	373	**361**
road-euroroad	260	287.6	255.6	262.2	**247**
email-EuAll	27 662	**26 000**	26 725	26 178	26 333
wordassociation-2011	36 168	37 513	**36 018**	36 084	35 923
Geometric Mean	5 722.45	6 255.31	5 655.97	**5 531.65**	5 581.64

Table B.10: GBDTs feature evaluation: Total cut sizes by selected features

Graph	KaFFPa Eco	Baseline	Statistical	Greedy	Heuristic
delaunay_n13	1 045	4 381	**3 783.2**	4 095	3 797.6
delaunay_n14	1 462	13 880	**10 065**	12 358.2	11 465.4
astro-ph	26 066	51 123	55 015.4	55 792.6	**54 831.8**
cond-mat	5 663	**15 293**	17 850.6	17 838.2	17 213.6
fe_4elt2	1 127	**6 322**	6 814.2	6 668	6 693.4
fe_sphere	1 959	4 532	4 929	4 843.4	**3 858**
road-minnesota	141	387	418	425.2	**351.6**
road-euroroad	100	**260**	271.6	268.2	264.8
email-EuAll	18 378	35 226	**28 046**	36 017.6	34 006.6
wordassociation-2011	31 266	36 186	35 915	**35 566**	36 223.4
Geometric Mean		6 671.60	6 542.20	6 880.00	**6 434.77**

Table B.11: SVM feature evaluation: Total cut sizes by feature group

Graph	Baseline	NC+PH	NC	PH
delaunay_n13	4 381	**3 783.2**	4 015.2 (+)	4 336.4 (+)
delaunay_n14	13 880	**10 065**	10 175.8 (+)	10 998 (+)
astro-ph	**51 123**	55 015.4	54 922 (-)	54 489 (-)
cond-mat	**15 293**	17 850.6	17 893.2 (+)	18 303.6 (+)
fe_4elt2	**6 322**	6 814.2	6 840 (+)	7 264.8 (+)
fe_sphere	**4 532**	4 929	4 906 (-)	8 569 (+)
road-minnesota	**387**	418	419.6 (+)	494.8 (+)
road-euroroad	**260**	271.6	268.2 (-)	287.4 (+)
email-EuAll	35 226	**28 046**	28 270 (+)	37 216.4 (+)
wordassociation-2011	36 168	**35 915**	35 915 (=)	37 964 (+)
Geometric Mean	6 671.60	**6 542.20**	6 587.78	7 541.30

Table B.12: SVM feature evaluation: Total cut sizes by statistical feature

Graph	**Baseline**	**NBC+EC+NB**	**EC+NB**
delaunay_n13	4 381	**4 095**	4 150 (+)
delaunay_n14	**13 880**	12 358.2	12 343.6 (-)
astro-ph	**51 123**	55 792.6	55 654.4 (-)
cond-mat	**15 293**	17 838.2	17 784.4 (-)
fe_4elt2	**6 322**	6 668	6 742.2 (+)
fe_sphere	**4 532**	4 843.4	4 765 (-)
road-minnesota	**387**	425.2	418.2 (-)
road-euroroad	260	268.2	270.2 (+)
email-EuAll	**35 226**	36 017.6	35 926.2 (-)
wordassociation-2011	36 186	35 566	35 611.6 (+)
Geometric Mean	**6 671.60**	6 880.00	6 873.79
Graph	**NBC**	**EC**	**NB**
delaunay_n13	4 239.2 (+)	4 216.6 (+)	4 142.4 (+)
delaunay_n14	12 258.6 (-)	11 992.2 (-)	11 607.8 (-)
astro-ph	55 781.8 (-)	55 852.8 (+)	56 036.6 (+)
cond-mat	17 842.6 (+)	17 815.6 (-)	17 855.8 (+)
fe_4elt2	7 052 (+)	6 790.2 (+)	7 038.4 (+)
fe_sphere	4 932 (+)	4 818.6 (+)	4 758 (-)
road-minnesota	391 (-)	399.6 (-)	390.6 (-)
road-euroroad	**253.6 (-)**	260.8 (-)	266.6 (-)
email-EuAll	37 156.2 (+)	36 124 (+)	37 221.6 (+)
wordassociation-2011	35 888.8 (+)	**35 537 (-)**	35 846.4 (+)
Geometric Mean	6 880.70	6 827.98	6 839.09

Table B.13: SVM feature evaluation: Total cut sizes by greedy feature

Graph	Baseline	Fennel+LDG	Fennel	LDG
delaunay_n13	4 381	**3 797.6**	3 895 (+)	4 302 (+)
delaunay_n14	13 880	11 465.4	11 915.8 (+)	**11 267 (-)**
astro-ph	**51 123**	54 831.8	54 280.8 (-)	55 422.6 (+)
cond-mat	**15 293**	17 213.6	16 831.8 (-)	17 938 (+)
fe_4elt2	**6 322**	6 693.4	6 627.4 (-)	6 925.6 (+)
fe_sphere	4 532	3 858	**3 685 (-)**	6 163.8 (+)
road-minnesota	387	351.6	**332.4 (-)**	407.6 (+)
road-euroroad	**260**	264.8	266 (+)	271 (+)
email-EuAll	35 226	**34 006.6**	34 069.2 (+)	36 838.2 (+)
wordassociation-2011	36 168	36 223.4	**36 159.4 (-)**	36 437.4 (+)
Geometric Mean	6 671.60	6 434.77	**6 386.04**	7 053.90

Table B.14: SVM feature evaluation: Total cut sizes by heuristic feature

Graph	Baseline	NC+PH	EC	Fennel	NC+PH+Fennel
delaunay_n13	4 381	**3 783.2**	4 216.6	3 895	3 864.2
delaunay_n14	13 880	**10 065**	11 992.2	11 915.8	10 221.8
astro-ph	**51 123**	55 015.4	55 852.8	54 280.8	55 021.2
cond-mat	**15 293**	17 850.6	17 815.6	16 831.8	17 861.2
fe_4elt2	**6 322**	6 814.2	6 790.2	6 627.4	6 870.6
fe_sphere	4 532	4 929	4 818.6	**3 685**	5 076.8
road-minnesota	387	418	399.6	**332.4**	424.8
road-euroroad	**260**	271.6	260.8	266	270.4
email-EuAll	35 226	**28 046**	36 124	34 069.2	28 084
wordassociation-2011	36 168	35 915	**35 537**	36 159.4	35 915
Geometric Mean	6 671.60	6 542.20	6 827.98	**6 386.04**	6 600.15

Table B.15: SVM feature evaluation: Total cut sizes by selected features

Graph	KaFFPa Eco	Baseline	Statistical	Greedy	Heuristic
delaunay_n13	1 045	4 381	**4 028.2**	4 220.8	4 240
delaunay_n14	1 462	7 989	**7 008.2**	7 589.4	7 575.6
astro-ph	26 066	**39 642**	40 276	40 258.8	40 336.8
cond-mat	5 663	**11 348**	12 003	11 946	11 761.8
fe_4elt2	1 127	**6 115**	6 757.4	6 827.2	6 203
fe_sphere	1 959	3 885	4 765	**3 802.6**	3 807.2
road-minnesota	141	387	474	**381.2**	384.8
road-euroroad	100	260	292.6	263	**254.4**
email-EuAll	18 378	**27 662**	28 077.2	28 443	28 229.2
wordassociation-2011	31 266	36 168	37 141.8	**36 004.4**	36 087
Geometric Mean		5 722.45	6 029.99	5 772.25	**5 695.04**

Table B.16: GraphSAGE feature evaluation: Total cut sizes by feature group

Graph	Baseline	NC+PH	NC	PH
delaunay_n13	4 381	**4 028.2**	5 247 (+)	4 286.2 (+)
delaunay_n14	7 989	**7 008.2**	9 672.4 (+)	7 514 (+)
astro-ph	**39 642**	40 276	42 281.8 (+)	40 860.2 (+)
cond-mat	**11 348**	12 003	13 600.4 (+)	11 933.4 (-)
fe_4elt2	**6 115**	6 757.4	7 749.8 (+)	6 911.8 (+)
fe_sphere	**3 885**	4 929	5 078 (+)	5 360.2 (+)
road-minnesota	**387**	474	662.4 (+)	491 (+)
road-euroroad	**260**	292.6	322.4 (+)	288.6 (-)
email-EuAll	**27 662**	28 077.2	27 970.6 (-)	28 499 (+)
wordassociation-2011	**36 168**	37 141.8	37 823.4 (+)	36 832 (-)
Geometric Mean	**5 722.45**	6 029.99	6 940.83	6 219.01

Table B.17: GraphSAGE feature evaluation: Total cut sizes by statistical feature

Graph	**Baseline**	**NBC+EC+NB**	**EC+NB**
delaunay_n13	4 381	4 220.8	4 163.4 (-)
delaunay_n14	7 989	7 589.4	**7 279.2 (-)**
astro-ph	**39 642**	40 258.8	40 298.6 (+)
cond-mat	**11 348**	11 946	11 886.8 (-)
fe_4elt2	**6 115**	6 827.2	6 714.6 (-)
fe_sphere	3 885	**3 802.6**	3 837.8 (+)
road-minnesota	**387**	381.2	**371 (-)**
road-euroroad	260	263	**258.2 (-)**
email-EuAll	**27 662**	28 443	28 386.6 (-)
wordassociation-2011	36 168	36 004.4	**35 884.2 (-)**
Geometric Mean	5 722.45	5 772.25	**5 704.71**
Graph	**NBC**	**EC**	**NB**
delaunay_n13	4 188 (-)	**4 107.4 (-)**	4 199.4 (-)
delaunay_n14	7 500.4 (-)	7 531.4 (-)	7 356 (-)
astro-ph	40 826.4 (+)	40 546.6 (+)	40 650.2 (+)
cond-mat	11 986.6 (+)	11 886.8 (-)	11 954.8 (+)
fe_4elt2	6 883.8 (+)	6 697.6 (-)	6 832.4 (+)
fe_sphere	3 874.6 (+)	3 820.2 (+)	3 858 (+)
road-minnesota	385.6 (+)	377.4 (-)	377.2 (-)
road-euroroad	261.2 (-)	271.2 (+)	259 (-)
email-EuAll	28 648 (+)	28 355.2 (-)	28 529 (+)
wordassociation-2011	35 952.8 (-)	36 020.6 (+)	36 016.8 (+)
Geometric Mean	5 792.58	5 755.38	5 753.12

Table B.18: GraphSAGE feature evaluation: Total cut sizes by greedy feature

Graph	Baseline	Fennel+LDG	Fennel	LDG
delaunay_n13	4 381	4 240	**4 205.6 (-)**	4 223.4 (-)
delaunay_n14	7 989	7 575.6	7 555.6 (-)	**7 062.2 (-)**
astro-ph	**39 642**	40 336.8	40 295.6 (-)	40 766.6 (+)
cond-mat	**11 348**	11 761.8	11 716.6 (-)	11 908 (+)
fe_4elt2	**6 115**	6 203	6 309.4 (+)	6 559.8 (+)
fe_sphere	3 885	3 807.2	**3 581.2 (-)**	4 056 (+)
road-minnesota	387	384.8	405 (+)	**368 (-)**
road-euroroad	260	**254.4**	255.2 (+)	260.8 (+)
email-EuAll	**27 662**	28 229.2	28 321.6 (+)	28 466.8 (+)
wordassociation-2011	36 168	36 087	**35 997.6 (-)**	36 224.4 (+)
Geometric Mean	5 722.45	5 695.04	**5 692.33**	5 729.58

Table B.19: GraphSAGE feature evaluation: Total cut sizes by heuristic feature

Graph	Baseline	NC+PH	EC+NB	Fennel	EC+NB+Fennel
delaunay_n13	4 381	4 028.2	4 163.4	4 205.6	**3 985.8**
delaunay_n14	7 989	7 008.2	7 279.2	7 555.6	**6 216**
astro-ph	**39 642**	40 276	40 298.6	40 295.6	40 876.4
cond-mat	**11 348**	12 003	11 886.8	11 716.6	11 973.4
fe_4elt2	**6 115**	6 757.4	6 714.6	6 309.4	6 573.6
fe_sphere	3 885	4 929	3 837.8	**3 581.2**	4 140.6
road-minnesota	387	474	**371**	405	403.8
road-euroroad	260	292.6	258.2	**255.2**	272.2
email-EuAll	**27 662**	28 077.2	28 386.6	28 321.6	27 866.6
wordassociation-2011	36 168	37 141.8	**35 884.2**	35 997.6	36 099.2
Geometric Mean	5 722.45	6 029.99	5 704.71	**5 692.33**	5 704.56

Table B.20: GraphSAGE feature evaluation: Total cut sizes by selected features

Graph	KaFFPa Eco	Baseline	Statistical	Greedy	Heuristic
delaunay_n13	1 045	4 381	**4 041.2**	4 228.6	4 275.2
delaunay_n14	1 462	7 989	**6 633**	7 674	7 406.4
astro-ph	26 066	**39 642**	40 543.4	40 450	40 379.6
cond-mat	5 663	**11 348**	12 003.6	11 895.4	11 876
fe_4elt2	1 127	**6 115**	6 733.4	6 817.8	6 237.8
fe_sphere	1 959	**3 885**	4 626.4	3 901.2	3 887.8
road-minnesota	141	387	480.8	**379.8**	448.8
road-euroroad	100	**260**	303.4	264.2	261.4
email-EuAll	18 378	27 662	28 200.8	28 464.8	**27 484**
wordassociation-2011	31 266	36 168	36 843	35 933.4	**36 138.2**
Geometric Mean		**5 722.45**	6 011.08	5 793.82	5 797.67

Table B.21: Partitioner feature evaluation: Total cut sizes by feature group

Graph	Baseline	NC+PH	NC	PH
delaunay_n13	4 381	**4 041.2**	5 123.8 (+)	4 275.8 (+)
delaunay_n14	7 989	**6 633**	9 829.2 (+)	7 296.8 (+)
astro-ph	**39 642**	40 543.4	42 960.8 (+)	40 982 (+)
cond-mat	**11 348**	12 003.6	13 546.8 (+)	12 118.6 (+)
fe_4elt2	**6 115**	6 733.4	7 646.6 (+)	6 983.2 (+)
fe_sphere	**3 885**	4 626.4	5 183.6 (+)	5 193.6 (+)
road-minnesota	**387**	480.8	643.8 (+)	520.2 (+)
road-euroroad	**260**	303.4	326.4 (+)	294.8 (-)
email-EuAll	**27 662**	28 200.8	27 848.4 (-)	28 366 (+)
wordassociation-2011	**36 168**	36 843	37 831.2 (+)	36 804.6 (+))
Geometric Mean	**5 722.45**	6 011.08	6 934.69	6 243.26

Table B.22: Partitioner feature evaluation: Total cut sizes by statistical feature

Graph	**Baseline**	**NBC+EC+NB**	**EC+NB**
delaunay_n13	4 381	4 228.6	4 239 (+)
delaunay_n14	7 989	7 674	7 537 (-)
astro-ph	**39 642**	40 450	40 407.4 (-)
cond-mat	**11 348**	11 895.4	11 899.6 (+)
fe_4elt2	**6 115**	6 817.8	6 725.8 (-)
fe_sphere	3 885	3 901.2	**3 812.8 (-)**
road-minnesota	387	**379.8**	390.2 (+)
road-euroroad	260	264.2	267.6 (+)
email-EuAll	**27 662**	28 464.8	28 467.2 (+)
wordassociation-2011	36 168	**35 933.4**	36 027.2 (+)
Geometric Mean	**5 722.45**	5 793.82	5 787.88
Graph	**NBC**	**EC**	**NB**
delaunay_n13	4 213.4 (-)	4 294.4 (+)	**4 166 (-)**
delaunay_n14	**7 384.2 (-)**	7 658.4 (-)	7 544.6 (-)
astro-ph	40 741.6 (+)	40 489.8 (+)	40 715 (+)
cond-mat	11 960.6 (+)	11 950.8 (+)	11 939.6 (+)
fe_4elt2	6 754.4 (-)	6 830 (+)	6 733.8 (-)
fe_sphere	3 876.8 (-)	3 857.4 (-)	3 944.2 (+)
road-minnesota	398 (+)	390.8 (+)	386.6 (+)
road-euroroad	266.8 (+)	271 (+)	**259.8 (-)**
email-EuAll	28 304.4 (-)	28 433.6 (-)	28 177 (-)
wordassociation-2011	36 007.4 (+)	36 002.2 (+)	36 203.2 (+)
Geometric Mean	5 798.42	5 831.21	5 779.45

Table B.23: Partitioner feature evaluation: Total cut sizes by greedy feature

Graph	Baseline	Fennel+LDG	Fennel	LDG
delaunay_n13	4 381	4 275.2	4 223.8 (-)	**4 014.6 (-)**
delaunay_n14	7 989	7 406.4	7 640 (+)	**7 400.6 (-)**
astro-ph	**39 642**	40 379.6	40 072.2 (-)	40 765.8 (+)
cond-mat	**11 348**	11 876	11 714.4 (-)	11 920.6 (+)
fe_4elt2	**6 115**	6 237.8	6 384.8 (+)	6 577 (+)
fe_sphere	3 885	3 887.8	**3 523.6 (-)**	4 017.6 (+)
road-minnesota	387	448.8	409 (-)	**376.2 (-)**
road-euroroad	**260**	261.4	262.2 (+)	262.6 (+)
email-EuAll	27 662	**27 484**	27 888 (+)	28 331.6 (+)
wordassociation-2011	36 168	**36 138.2**	36 225.2 (+)	36 289.6 (+)
Geometric Mean	5 722.45	5 797.67	**5 711.20**	5 738.87

Table B.24: Partitioner feature evaluation: Total cut sizes by heuristic feature

Graph	Baseline	NC+PH	NB	Fennel	NB+Fennel
delaunay_n13	4 381	**4 041.2**	4 166	4 223.8	4 257.4
delaunay_n14	7 989	**6 633**	7 544.6	7 640	7 513.6
astro-ph	**39 642**	40 543.4	40 715	40 072.2	40 148.8
cond-mat	**11 348**	12 003.6	11 939.6	11 714.4	11 836.4
fe_4elt2	**6 115**	6 733.4	6 733.8	6 384.8	6 555.4
fe_sphere	3 885	4 626.4	3 944.2	**3 523.6**	3 657.6
road-minnesota	387	480.8	**386.6**	409	398.2
road-euroroad	260	303.4	**259.8**	262.2	264.4
email-EuAll	**27 662**	28 200.8	28 177	27 888	27 920.2
wordassociation-2011	**36 168**	36 843	36 203.2	36 225.2	36 169.8
Geometric Mean	5 722.45	6 011.08	5 779.45	**5 711.20**	5 738.93

Table B.25: Partitioner feature evaluation: Total cut sizes by selected features

APPENDIX C

Generalization

C.1 Instance Generalization

C.1.1 Perfectly Balanced Partitioning

Graph	Baseline	Logistic Regression	GBDTs
delaunay_n15	6 987	**363**	5 357
rgg_n_2_15_s0	1 673	**376**	776
cond-mat-2003	12 148	13 198	12 985.8
cond-mat-2005	18 620	20 489	**17 062.8**
bcsstk30	23 960	6 901	**6 607**
cs4	4 756	**1 334**	3 589.8
as-22july06	8 409	**8 365**	**8 365**
soc-Slashdot0902	80 287	**56 120**	**56 120**
Geometric Mean	10 928.2	**4 470.4**	7 522.64
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	16 686	5 475.4	4 729.6
rgg_n_2_15_s0	994	1 159.6	999
cond-mat-2003	35 670	11 714.4	**11 456.4**
cond-mat-2005	58 817	17 789.6	17 183.8
bcsstk30	45 240	17 601.4	6 710.4
cs4	15 060	3 006.6	1 907.6
as-22july06	13 839	8 600	8 643.8
soc-Slashdot0902	184 804	79 995.4	104 271.8
Geometric Mean	22 276.07	9 127.45	7 565.42

Table C.1: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 2$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	10 754	**5 600**	9 509.8
rgg_n_2_15_s0	3 073	**1 813**	2 575
cond-mat-2003	22 145	22 907	22 534.6
cond-mat-2005	**33 213**	35 176	34 100.8
bcsstk30	79 918	**41 587**	53 289.4
cs4	7 753	**6 590**	7 143
as-22july06	13 320	13 466	**12 780**
soc-Slashdot0902	167 452	**136 941**	161 205.4
Geometric Mean	20 690.84	**15 923.99**	18 669.14
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	19 932	8 302.8	7 622
rgg_n_2_15_s0	2 966	2 655.2	2 745.2
cond-mat-2003	52 569	21 896.8	**21 782.4**
cond-mat-2005	85 162	33 503.2	33 821.6
bcsstk30	71 501	49 732.8	48 621.8
cs4	17 080.6	6 727.8	6 721
as-22july06	21 844	13 278.4	13 303
soc-Slashdot0902	249 652	171 965.8	176 685
Geometric Mean	33 944.74	18 259.4	18 162.25

Table C.2: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 4$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	13 111	**10 004**	10 712
rgg_n_2_15_s0	4 860	4 046	4 516.8
cond-mat-2003	**29 467**	30 482	29 734.2
cond-mat-2005	**45 073**	46 453	45 100.4
bcsstk30	124 795	90 787	**77 782**
cs4	9 184	10 506	8 576.2
as-22july06	17 669	18 140	17 518.6
soc-Slashdot0902	221 958	**208 446.6**	220 159.6
Geometric Mean	28 023.96	25 966.82	**25 282.42**
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	36 438	11 267.6	11 876
rgg_n_2_15_s0	4 535	4 405.4	**3 780.2**
cond-mat-2003	61 699	29 985.2	29 856.4
cond-mat-2005	99 967	45 112.2	45 605.4
bcsstk30	134 160	87 753.6	96 837.4
cs4	23 760.2	8 897.2	**8 561**
as-22july06	27 451	**17 503.8**	17 588.6
soc-Slashdot0902	288 891	225 028.6	227 224
Geometric Mean	47 465.6	25 963.5	25 900.7

Table C.3: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 8$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	14 885	**12 610**	13 373
rgg_n_2_15_s0	6 654	6 845	6 842
cond-mat-2003	**35 365**	36 320.2	35 687.4
cond-mat-2005	**54 807**	55 921.8	55 041
bcsstk30	163 903	**154 038**	155 017.6
cs4	10 397	11 247.4	**10 083**
as-22july06	20 441	20 397	**20 387.8**
soc-Slashdot0902	260 967	264 531.4	**256 390.4**
Geometric Mean	33 902.21	33 637.03	33 178.52
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	37 174.4	13 098.2	13 750.2
rgg_n_2_15_s0	8 041	**6 091.4**	6 682.2
cond-mat-2003	69 904.6	35 502.6	35 875.4
cond-mat-2005	113 545.6	55 167	55 347
bcsstk30	213 668	157 004.8	154 471.2
cs4	25 811.4	10 246.4	10 342
as-22july06	29 801.8	20 516.2	20 484.2
soc-Slashdot0902	325 313.6	263 119	263 099.6
Geometric Mean	57 931.63	**32 853.01**	33 459.07

Table C.4: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$, $\epsilon = 0.0$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

C.1.2 Imbalanced Partitioning

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	6 867.6	10 584	**5 354**
rgg_n_2_15_s0	1 459.6	918	722.2
cond-mat-2003	12 728	12 423	13 012.8
cond-mat-2005	19 724	22 266	18 829.4
bcsstk30	31 807.8	7 357	**6 251**
cs4	4 810.2	**1 668**	3 462
as-22july06	8 676.8	8 561	8 844
soc-Slashdot0902	116 207.2	104 998	104 998
Geometric Mean	11 847.74	8 590.05	8 127.48
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	43 048	5 625.8	4 376.8
rgg_n_2_15_s0	957	1 177.2	**714**
cond-mat-2003	26 693	**12 069.6**	12 289.2
cond-mat-2005	44 338	**18 048.4**	18 369.2
bcsstk30	46 250	17 241	6 476.4
cs4	14 835	3 497.6	2 044.4
as-22july06	14 392	8 770.6	**8 506**
soc-Slashdot0902	**104 646**	111 583.8	105 536
Geometric Mean	21 766.33	9 800.9	**7 335.62**

Table C.5: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 2$, $\epsilon = 0.03$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	10 727.6	**6 451**	9 715
rgg_n_2_15_s0	2 716	**2 072**	2 403
cond-mat-2003	25 751.8	26 703	25 932
cond-mat-2005	42 807.4	42 625	42 664.4
bcsstk30	72 718.8	45 552	37 079.8
cs4	7 702.4	6 786	7 029
as-22july06	14 026.8	14 016	14 456.2
soc-Slashdot0902	223 495.6	223 368	**217 798.6**
Geometric Mean	22 075.34	18 663.84	19 533.49
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	20 118	8 832.2	7 870.6
rgg_n_2_15_s0	2 584.6	2 601.6	2 390.4
cond-mat-2003	47 990	25 373.2	**25 349.2**
cond-mat-2005	77 955	42 684.4	**42 445.4**
bcsstk30	84 488	54 373	**34 943.8**
cs4	20 013	6 912.6	**6 347**
as-22july06	22 988	**13 754**	14 080.2
soc-Slashdot0902	247 521	221 448.2	225 957.2
Geometric Mean	34 199.85	20 269.12	**18 593.27**

Table C.6: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 4$, $\epsilon = 0.03$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	13 132.4	**8 633**	11 424
rgg_n_2_15_s0	4 551	4 386.4	4 086
cond-mat-2003	**33 672**	34 606	34 050.2
cond-mat-2005	**55 079.4**	55 954	55 384.6
bcsstk30	111 467.8	80 717	78 151
cs4	9 180	10 502	**8 604.4**
as-22july06	19 295.2	19 388	19 479.8
soc-Slashdot0902	278 370.4	279 372	**275 860.6**
Geometric Mean	29 723.28	27 599.47	27 404.34
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	36 239.4	11 203.4	11 908.4
rgg_n_2_15_s0	4 480	3 959.4	**3 865.2**
cond-mat-2003	61 482	34 050.8	34 091.4
cond-mat-2005	103 566	55 257.6	55 290.6
bcsstk30	139 758	91 235.8	**75 933**
cs4	23 836.8	8 878.2	8 719
as-22july06	27 392	19 432	**19 096**
soc-Slashdot0902	304 741	278 536.2	279 333.4
Geometric Mean	48 119.98	27 888.39	**27 276.3**

Table C.7: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 8$, $\epsilon = 0.03$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	14 745.2	**12 734**	13 252.6
rgg_n_2_15_s0	6 409.6	6 933.2	6 386
cond-mat-2003	**37 880.8**	39 076	38 308.8
cond-mat-2005	**61 528**	62 753	61 889
bcsstk30	163 109.4	170 763	**156 210.8**
cs4	10 274	11 106.8	**9 964**
as-22july06	**22 390.4**	22 522	22 528
soc-Slashdot0902	**301 542.6**	302 644	301 752.4
Geometric Mean	35 446.66	35 964.22	**34 743.22**
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	36 477.2	13 081	13 546.2
rgg_n_2_15_s0	7 816.8	6 389.6	**6 157.2**
cond-mat-2003	71 676.8	38 237.2	38 397
cond-mat-2005	114 601.2	61 915.4	62 154.6
bcsstk30	216 431	162 122.8	156 981.8
cs4	25 712.2	10 210	10 279.6
as-22july06	31 865	22 459.8	22 425
soc-Slashdot0902	330 137.2	302 307.4	302 671.4
Geometric Mean	58 497.28	34 945.44	34 858.81

Table C.8: Instance generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$, $\epsilon = 0.03$ and using a uniformly sampled 80%/20% train-test-split. Each experiment on each graph is repeated five times using different random seeds. The reported total cut sizes are based on the true block labels of the training nodes and the predicted block labels of the testing nodes.

C.2 Group Generalization

C.2.1 Perfectly Balanced Partitioning

Graph	Baseline	Logistic Regression	GBDTs
delaunay_n15	17 530	25 457	**9 556.4**
rgg_n_2_15_s0	3 274	**768**	2 235.8
cond-mat-2003	**13 347**	35 252	16 538.6
cond-mat-2005	**20 136**	50 707	21 629.8
bcsstk30	78 189	**24 213**	56 122.2
cs4	9 211	13 122	**9 148.8**
as-22july06	**14 210**	18 988	18 947
soc-Slashdot0902	155 577	**105 974**	179 795.4
Geometric Mean	19 896.48	19 662.71	**18 440.05**
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	25 457	17 576.8	17 850.4
rgg_n_2_15_s0	**768**	2 071.6	2 129.8
cond-mat-2003	35 252	16 907	18 770
cond-mat-2005	50 707	27 665.4	27 912
bcsstk30	**24 213**	53 028	87 532.6
cs4	13 122	15 162.6	16 553.4
as-22july06	18 988	18 839.2	19 199
soc-Slashdot0902	**105 974**	152 360.4	143 920.2
Geometric Mean	19 662.71	21 101.23	23 052.2

Table C.9: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 2$ and $\epsilon = 0.0$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	27 786	37 736	**21 992.6**
rgg_n_2_15_s0	6 087	**1 473**	5 272
cond-mat-2003	**26 502**	58 229	29 529.2
cond-mat-2005	**42 849**	87 455	43 119.2
bcsstk30	139 297	**51 676**	85 477.6
cs4	**14 419**	23 677	16 336
as-22july06	**19 736**	29 278	27 688.8
soc-Slashdot0902	231 569	223 789	256 666.8
Geometric Mean	33 945.57	35 033.6	**33 168.88**
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	37 736	29 673.6	27 396.8
rgg_n_2_15_s0	**1 473**	3 775.4	3 517.4
cond-mat-2003	58 229	33 782.8	29 986
cond-mat-2005	87 455	55 695.6	48 319.2
bcsstk30	**51 676**	94 056.8	96 381.4
cs4	23 677	21 081.2	18 747.4
as-22july06	29 278	27 228	26 902.8
soc-Slashdot0902	223 789	**220 254.4**	232 738.8
Geometric Mean	35 033.6	35 471.9	33 482.67

Table C.10: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 4$ and $\epsilon = 0.0$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	33 059	39 697	**22 768**
rgg_n_2_15_s0	7 438	**2 735**	6 847.8
cond-mat-2003	**32 397**	74 384	42 076.8
cond-mat-2005	**51 590**	112 236	61 329.6
bcsstk30	210 404	**96 640**	126 999
cs4	**17 810**	30 512	22 829.2
as-22july06	**26 588**	36 038	33 788.6
soc-Slashdot0902	**279 331**	298 526	307 148.6
Geometric Mean	**42 883.71**	48 121.28	43 189.17
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	39 697	34 553.4	35 046.6
rgg_n_2_15_s0	**2 735**	5 192	5 395
cond-mat-2003	74 384	43 041.2	43 242
cond-mat-2005	112 236	70 470.2	68 268.8
bcsstk30	**96 640**	130 786.4	121 727
cs4	30 512	25 600.4	22 609.4
as-22july06	36 038	32 317	33 780.4
soc-Slashdot0902	298 526	287 199.4	283 881.4
Geometric Mean	48 121.28	45 031.6	44 261.93

Table C.11: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 8$ and $\epsilon = 0.0$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	35 771	40 998	**27 674.4**
rgg_n_2_15_s0	9 490	**5 514**	11 352
cond-mat-2003	**35 394**	82 643	43 333.8
cond-mat-2005	**59 755**	125 389	68 399
bcsstk30	248 636	**151 155**	193 439.8
cs4	**19 592**	34 408	25 343
as-22july06	**27 239**	40 373	36 861.2
soc-Slashdot0902	**304 295**	335 363	333 890.4
Geometric Mean	**48 170.23**	59 864.99	52 323.95
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	40 998	38 304.2	39 574
rgg_n_2_15_s0	**5 514**	7 474.8	7 494.8
cond-mat-2003	82 643	47 670.2	51 856.2
cond-mat-2005	125 389	75 793.4	81 159.2
bcsstk30	**151 155**	187 483.6	197 673.4
cs4	34 408	27 818.6	26 419
as-22july06	40 373	36 548.6	38 776.8
soc-Slashdot0902	335 363	326 692.6	322 893.2
Geometric Mean	59 864.99	53 226.65	54 824.6

Table C.12: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$ and $\epsilon = 0.0$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

C.2.2 Imbalanced Partitioning

Graph	Baseline	Logistic Regression	GBDTs
delaunay_n15	17 415.6	24 646	**6 784.8**
rgg_n_2_15_s0	3 584.4	**972**	2 223.2
cond-mat-2003	**15 961.4**	34 161	16 512.6
cond-mat-2005	**24 661**	49 502	24 692
bcsstk30	77 855	**29 872**	60 460.2
cs4	9 125.4	12 901	**8 703.4**
as-22july06	**13 578**	18 431	18 406.4
soc-Slashdot0902	155 251.4	**100 330**	180 139.6
Geometric Mean	20 928.15	20 302.72	**17 940.23**
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	24 646	18 697.6	18 020.6
rgg_n_2_15_s0	**972**	1 975.4	2 037.2
cond-mat-2003	34 161	18 121	17 691.6
cond-mat-2005	49 502	29 982.2	26 385.2
bcsstk30	**29 872**	51 235.4	39 721.4
cs4	12 901	11 402.4	14 038.6
as-22july06	18 431	17 911.4	18 302.2
soc-Slashdot0902	**100 330**	140 275.8	157 519.42
Geometric Mean	20 302.72	20 353.81	20 184.34

Table C.13: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 2$ and $\epsilon = 0.03$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	27 406	37 394	**18 127.8**
rgg_n_2_15_s0	5 524.4	**1 706**	3 941
cond-mat-2003	**26 513.6**	57 390	29 888.4
cond-mat-2005	**40 727.2**	86 315	48 065.6
bcsstk30	134 538.6	**52 547**	82 804.8
cs4	**14 535.8**	23 651	16 435
as-22july06	**19 347.8**	28 914	27 089.6
soc-Slashdot0902	234 423.2	**219 549**	250 297.2
Geometric Mean	33 126.48	35 448.07	**31 408.45**
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	37 394	29 799.8	27 304.8
rgg_n_2_15_s0	**1 706**	3 682.2	3 938.6
cond-mat-2003	57 390	32 303	28 597.2
cond-mat-2005	86 315	49 722.2	46 608.6
bcsstk30	**52 547**	94 169.4	90 816.8
cs4	23 651	20 405.4	19 231.4
as-22july06	28 914	26 989.2	26 166.4
soc-Slashdot0902	**219 549**	219 851.2	231 488.4
Geometric Mean	35 448.07	34 505.31	33 312.14

Table C.14: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 4$ and $\epsilon = 0.03$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	32 888.6	39 791	**23 572**
rgg_n_2_15_s0	7 002.6	**2 992**	7 176.6
cond-mat-2003	**32 714**	73 718	42 050.4
cond-mat-2005	**51 238.8**	111 757	59 780.2
bcsstk30	207 217.4	**95 677**	144 442
cs4	**17 733.4**	30 467	22 529.4
as-22july06	**26 259.6**	35 697	33 639.6
soc-Slashdot0902	**277 672**	296 625	304 595.4
Geometric Mean	42 348.15	48 432.17	44 051.17
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	39 791	34 509	34 469.6
rgg_n_2_15_s0	**2 992**	5 273.4	4 954.4
cond-mat-2003	73 718	42 589.6	38 954.6
cond-mat-2005	111 757	67 666.2	63 583.4
bcsstk30	**95 677**	126 124	115 924
cs4	30 467	25 004.4	21 721.8
as-22july06	35 697	32 012	32 300.2
soc-Slashdot0902	296 625	281 752	284 965.4
Geometric Mean	48 432.17	44 331.41	**42 065.35**

Table C.15: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 8$ and $\epsilon = 0.03$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	35 746.6	41 197	**27 549.8**
rgg_n_2_15_s0	8 751	**5 951**	11 275
cond-mat-2003	**35 907.4**	82 382	42 190.6
cond-mat-2005	**58 695**	125 207	66 560
bcsstk30	239 716	**137 929**	193 430
cs4	**19 496.4**	34 449	24 362
as-22july06	**27 007.6**	40 115	37 146.6
soc-Slashdot0902	**303 685.4**	334 164	334 461.2
Geometric Mean	**47 351.08**	59 686.61	51 703.59
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	41 197	37 303.8	39 786
rgg_n_2_15_s0	**5 951**	6 865	7 170.2
cond-mat-2003	82 382	47 690.4	51 270
cond-mat-2005	125 207	77 355.8	81 098.4
bcsstk30	**137 929**	176 805.6	186 042.8
cs4	34 449	27 678.4	26 406
as-22july06	40 115	36 199.2	37 824.6
soc-Slashdot0902	334 164	325 541.8	322 680.4
Geometric Mean	59 686.61	52 123.37	53 888.98

Table C.16: Group generalization evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$ and $\epsilon = 0.03$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

APPENDIX D

Imbalance

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	35 771	40 998	**27 674.4**
rgg_n_2_15_s0	9 490	**5 514**	11 352
cond-mat-2003	**35 394**	82 643	43 333.8
cond-mat-2005	**59 755**	125 389	68 399
bcsstk30	248 636	**151 155**	193 439.8
cs4	**19 592**	34 408	25 343
as-22july06	**27 239**	40 373	36 861.2
soc-Slashdot0902	**304 295**	335 363	333 890.4
Geometric Mean	**48 170.23**	59 864.99	52 323.95
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	40 998	38 304.2	39 574
rgg_n_2_15_s0	**5 514**	7 474.8	7 494.8
cond-mat-2003	82 643	47 670.2	51 856.2
cond-mat-2005	125 389	75 793.4	81 159.2
bcsstk30	**151 155**	187 483.6	197 673.4
cs4	34 408	27 818.6	26 419
as-22july06	40 373	36 548.6	38 776.8
soc-Slashdot0902	335 363	326 692.6	322 893.2
Geometric Mean	59 864.99	53 226.65	54 824.6

Table D.1: Imbalance evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$ and $\epsilon = 0.0$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	Baseline	Logistic Regression	GBDTs
delaunay_n15	35 746.6	41 197	**27 549.8**
rgg_n_2_15_s0	8 751	**5 951**	11 275
cond-mat-2003	**35 907.4**	82 382	42 190.6
cond-mat-2005	**58 695**	125 207	66 560
bcsstk30	239 716	**137 929**	193 430
cs4	**19 496.4**	34 449	24 362
as-22july06	**27 007.6**	40 115	37 146.6
soc-Slashdot0902	**303 685.4**	334 164	334 461.2
Geometric Mean	**47 351.08**	59 686.61	51 703.59
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	41 197	37 303.8	39 786
rgg_n_2_15_s0	**5 951**	6 865	7 170.2
cond-mat-2003	82 382	47 690.4	51 270
cond-mat-2005	125 207	77 355.8	81 098.4
bcsstk30	**137 929**	176 805.6	186 042.8
cs4	34 449	27 678.4	26 406
as-22july06	40 115	36 199.2	37 824.6
soc-Slashdot0902	334 164	325 541.8	322 680.4
Geometric Mean	59 686.61	52 123.37	53 888.98

Table D.2: Imbalance evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$ and $\epsilon = 0.03$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	35 745.8	41 036	**27 099.6**
rgg_n_2_15_s0	8 549.2	**5 268**	10 208
cond-mat-2003	**35 721.8**	82 091	41 346.6
cond-mat-2005	**58 570.8**	124 603	68 592
bcsstk30	238 486.2	**144 226**	181 857.4
cs4	**19 585.4**	34 163	26 385
as-22july06	**26 699.4**	39 784	37 163.4
soc-Slashdot0902	**303 043.8**	332 505	333 678.2
Geometric Mean	**47 086.45**	58 863.03	51 126.2
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	41 036	38 564	39 898
rgg_n_2_15_s0	**5 268**	6 474.2	7 144.6
cond-mat-2003	82 091	45 544.4	49 610.8
cond-mat-2005	124 603	73 461.6	78 226.4
bcsstk30	**144 226**	174 297.8	179 464.8
cs4	34 163	27 989	26 714.2
as-22july06	39 784	35 828.4	38 641.6
soc-Slashdot0902	332 505	320 451.6	319 172.4
Geometric Mean	58 863.03	51 140.69	53 328.28

Table D.3: Imbalance evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$ and $\epsilon = 0.07$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

Graph	**Baseline**	**Logistic Regression**	**GBDTs**
delaunay_n15	35 774.4	41 050	**26 263**
rgg_n_2_15_s0	8 602	**5 457**	11 492
cond-mat-2003	**35 876.6**	81 981	41 189.6
cond-mat-2005	**58 117.2**	123 894	66 353
bcsstk30	236 260.6	**137 130**	193 564.2
cs4	**19 564.4**	34 147	23 342
as-22july06	**26 570.2**	39 769	37 241
soc-Slashdot0902	**306 495.2**	331 1764	331 810
Geometric Mean	47 083.69	58 666.46	51 039.66
Graph	**SVM**	**GraphSAGE**	**Partitioner**
delaunay_n15	41 050	38 284.8	39 439
rgg_n_2_15_s0	**5 457**	6 413.8	7 129.8
cond-mat-2003	81 981	47 180	48 315.8
cond-mat-2005	123 894	75 922.4	79 129.8
bcsstk30	**137 130**	175 543.6	177 738.6
cs4	34 147	27 240.8	26 634.4
as-22july06	39 769	36 688	37 219.6
soc-Slashdot0902	331 176	320 883	319 424.2
Geometric Mean	58 666.46	51 504.26	52 811.02

Table D.4: Imbalance evaluation: Total cut sizes by model. Experiments run on the evaluation dataset with $k = 16$ and $\epsilon = 0.1$. Each experiment on each graph is repeated five times using different random seeds. For every group each model was retrained using all graph instances from the same group in the tuning dataset.

APPENDIX E

Running Times

K	Baseline	Logistic Regression	GBDTs
2	- / 9.58	3.42 / 53.62	4.16 / 143.60
4	- / 12.27	3.69 / 56.68	5.36 / 152.64
8	- / 12.84	4.43 / 64.61	4.34 / 157.08
16	- / 12.93	11.75 / 77.67	7.05 / 162.03
Geometric Mean	- / 11.82	5.06 / 62.49	5.12 / 153.69
K	**SVM**	**GraphSAGE**	**Partitioner**
2	5.85 / 67.86	321.81 / 217.52	326.22 / 235.30
4	5.16 / 62.38	343.38 / 294.68	335.58 / 252.65
8	5.37 / 71.13	324.96 / 263.83	334.38 / 274.76
16	6.42 / 90.10	340.03 / 296.41	363.70 / 314.99
Geometric Mean	5.68 / 72.17	332.41 / 266.08	339.68 / 267.82

Table E.1: Running time evaluation: Running times for training and prediction by model. Experiments run on delaunay_n15 with $\epsilon = 0.03$. Each experiment is repeated five times using different random seeds. The entries have the format "training time/prediction time" and are measured in seconds.

K	Baseline	Logistic Regression	GBDTs
2	- / 7.68	3.17 / 39.63	4.40 / 107.06
4	- / 8.93	3.41 / 41.52	4.21 / 105.48
8	- / 9.17	3.81 / 46.63	5.05 / 108.07
16	- / 9.15	4.91 / 54.98	20.56 / 119.52
Geometric Mean	- / 8.71	3.77 / 45.32	6.62 / 109.90
K	**SVM**	**GraphSAGE**	**Partitioner**
2	5.04 / 42.88	363.17 / 128.69	362.35 / 140.09
4	7.97 / 48.33	398.74 / 155.22	378.07 / 144.94
8	7.76 / 54.86	376.49 / 141.28	385.10 / 153.00
16	8.78 / 67.12	398.10 / 155.72	422.20 / 171.53
Geometric Mean	7.23 / 52.55	383.83 / 144.79	386.32 / 151.93

Table E.2: Running time evaluation: Running times for training and prediction by model. Experiments run on cond-mat-2003 with $\epsilon = 0.03$. Each experiment is repeated five times using different random seeds. The entries have the format "training time/prediction time" and are measured in seconds.

K	Baseline	Logistic Regression	GBDTs
2	- / 4.59	1.73 / 23.25	2.04 / 62.63
4	- / 5.05	1.87 / 24.61	2.73 / 63.05
8	- / 5.25	2.15 / 27.94	4.21 / 65.13
16	- / 5.28	3.28 / 33.48	4.31 / 66.29
Geometric Mean	- / 5.03	2.19 / 27.05	3.17 / 64.26
K	**SVM**	**GraphSAGE**	**Partitioner**
2	4.47 / 25.34	165.88 / 54.60	170.76 / 59.09
4	6.64 / 27.33	174.12 / 64.76	178.75 / 61.57
8	12.05 / 33.73	165.6 / 60.01	180.86 / 64.4
16	15.40 / 43.16	172.39 / 64.72	189.02 / 70.24
Geometric Mean	8.61 / 30.49	169.45 / 60.88	179.73 / 63.69

Table E.3: Running time evaluation: Running times for training and prediction by model. Experiments run on cs4 with $\epsilon = 0.03$. Each experiment is repeated five times using different random seeds. The entries have the format "training time/prediction time" and are measured in seconds.

K	Baseline	Logistic Regression	GBDTs
2	- / 4.57	1.60 / 21.31	1.92 / 57.16
4	- / 4.88	1.76 / 22.27	4.58 / 57.40
8	- / 5.12	2.01 / 25.01	3.09 / 58.57
16	- / 5.05	3.00 / 29.52	14.55 / 64.46
Geometric Mean	- / 4.90	2.03 / 24.33	4.46 / 59.33
K	**SVM**	**GraphSAGE**	**Partitioner**
2	4.79 / 23.93	156.81 / 79.45	155.18 / 82.76
4	9.64 / 28.25	162.27 / 97.17	159.74 / 87.22
8	6.72 / 32.16	157.08 / 87.25	158.26 / 92.47
16	9.44 / 39.75	162.66 / 96.42	168.57 / 104.00
Geometric Mean	7.36 / 30.49	159.68 / 89.77	160.36 / 91.28

Table E.4: Running time evaluation: Running times for training and prediction by model. Experiments run on as-22july06 with $\epsilon = 0.03$. Each experiment is repeated five times using different random seeds. The entries have the format "training time/prediction time" and are measured in seconds.