Karlsruhe Institute of Technology

Bachelor Thesis

Machine-Learning based Hypergraph Pruning for Partitioning

Tobias Fuchs

Abgabedatum: 31.07.2020

Supervisors: Prof. Dr. Peter Sanders
M.Sc. Tobias Heuer
Dr. Christian Schulz, Privatdoz.[1]
M.Sc. Daniel Seemaier

Institute of Theoretical Informatics, Algorithmics
Department of Informatics
Karlsruhe Institute of Technology

[1]Research Group Theory and Applications of Algorithms, Department of Informatics, University of Vienna

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Mühlacker, den 31.07.2020

Tobias Fuchs

Abstract

Hypergraph partitioning is a useful tool for improving electrical circuit designs and accelerating sparse matrix vector multiplications for example. A hypergraph is a generalisation of a graph in which an edge may contain more than two nodes. In hypergraph partitioning, vertices should be distributed to a fixed number of blocks while maintaining a balance constraint on the size of the blocks as well as minimising an objective function between those. Due to the $\mathcal{NP}$-hardness of hypergraph partitioning, heuristics are used to deal with otherwise intractable problem instances. One of the most important meta-heuristics is the multi-level paradigm. It is three-layered consisting of the coarsening, initial partitioning and refinement phase. This work focuses mainly on the coarsening phase since the selection of a proper rating function is still an interesting avenue of research.

We propose a machine-learning based approach that uses a logistic regression model to estimate the likelihood of two adjacent hypernodes to end up in the same block of a partition. Sample data consists of feature vectors that are computed using global statistics on the hypergraph as well as local information on the adjacent hypernodes considered. Thereby, coarsening rating functions used in other well-established partitioners are also used as feature values. The prediction of the trained model is between 97% and 99% accurate if the prediction is that a pair of particular hypernodes belong to the same block. Additionally, we predict between 70% and 73% of all considered hypernode pairs to belong to the same block of a partition.

The trained model has been embedded into a coarsening algorithm. After this algorithm is applied, we use KaHyPar-CA to calculate a partition on the coarse hypergraph followed by refinement to the original instance. Our coarsening algorithm contracts on average about 24% of all pins among all different hypergraph classes. Although, the prediction making consumes more time than calculating only a single rating function in coarsening and the final partitions are on average slightly worse than the results of a state-of-the-art partitioner (KaHyPar-CA). However, the trained model's weights reveal interesting insights about the rating function's importances.

Zusammenfassung

Hypergraph-Partitionierung ist ein nützliches Werkzeug zur Modellierung von Problemen verschiedener Domänen. Das Verbessern von elektrischen Schaltplänen, sowie die Beschleunigung der Multiplikation von dünnbesetzten Matrizen mit Vektoren, sind mögliche Anwendungen. Ein Hypergraph ist eine Verallgemeinerung eines Graphen in dem eine Kante mehr als zwei Knoten enthalten kann. Bei der Hypergraph-Partitionierung wird angestrebt Hyperknoten auf eine feste Anzahl von Blöcken zu verteilen, sodass die Größen der Blöcke möglichst gleich sind. Gleichzeitig soll eine Zielfunktion über den Hyperkanten, die Hyperknoten mehrerer Blöcke beinhalten, minimiert werden. Da Hypergraph-Partitionierung ein $\mathcal{NP}$-schweres Problem ist, werden Heuristiken verwendet um mit ansonsten unlösbaren Probleminstanzen umzugehen. Eine der wichtigsten Meta-Heuristiken ist das Multilevel-Paradigma. Es besteht aus drei Schritten: der Vergröberungs-, der Partitionierungs- und der Verfeinerungsphase. Diese Arbeit konzentriert sich hauptsächlich auf die Vergröberungsphase, da die Auswahl einer geeigneten Bewertungsfunktion in dieser immer noch ein interessanter Forschungsweg ist.

In dieser Arbeit wird ein auf maschinellem Lernen basierender Ansatz vorgestellt, der eine logistische Regression verwendet, um die Wahrscheinlichkeit abzuschätzen, dass zwei benachbarte Hyperknoten im gleichen Block einer Partition landen. Die Eingabedaten bestehen dabei aus Merkmalsvektoren, die unter Verwendung globaler Statistiken über den Hypergraphen sowie lokaler Informationen über die betrachteten, benachbarten Hyperknoten berechnet werden. Dabei werden auch Bewertungsfunktionen, die in anderen, etablierten Partitionierern verwendet werden, als Merkmalswerte verwendet. Die Vorhersage des trainierten Modells ist zwischen 97% und 99% genau, wenn vorhergesagt wird, dass zwei benachbarte Hyperknoten zum selben Block gehören. Außerdem werden zwischen 70% und 73% aller betrachteten Hyperknotenpaare mit dieser Vorhersage belegt.

Das trainierte Modell wurde in einen Vergröberungsalgorithmus eingebettet. Nach dessen Anwendung wird der Partitionierer KaHyPar-CA verwendet, um eine Partition auf dem reduzierten Hypergraphen zu berechnen, gefolgt von einer Verfeinerung zur ursprünglichen Instanz. Der vorgestellte Vergröberungsalgorithmus kontrahiert durchschnittlich etwa 24% aller Pins über alle verschiedenen Hypergraphenklassen hinweg. Obwohl die Berechnung der Vorhersage mehr Zeit in Anspruch nimmt, sind die berechneten Partitionen im Durchschnitt etwas schlechter als die eines modernen Partitionierers (KaHyPar-CA). Die Gewichte des trainierten Modells offenbaren jedoch interessante Erkenntnisse über die Wichtigkeit verschiedener Bewertungsfunktionen in der Vergröberungsphase.

Danksagungen

In erster Linie möchte ich mich natürlich bei meinen Betreuern Christian, Daniel und Tobias bedanken, die mich stets mit guten Ideen und Ratschlägen bei meiner Arbeit unterstützten. Außerdem gehört mein Dank meinen Freunden und meiner Familie, die stets an meiner Seite standen, auch wenn ich mal nur weniger Zeit mit ihnen verbringen konnte.

Contents

1. Introduction

1.1. Motivation

Hypergraphs are a generalisation of graphs that may have more than two nodes per edge. They are useful for modelling for example group chats in social networks [77] or connectivity of electrical components in circuits [40]. In hypergraph partitioning, vertices should be distributed to a fixed number of blocks while maintaining a balance constraint on the size of the blocks as well as minimising an objective function between those. Applications of hypergraph partitioning include modelling group chats in social networks using partitioning to overcome scaling issues [77]. Also, improving electrical circuit designs [40], optimising transportations on road networks [76] as well as solving SAT problems [18] and accelerating sparse matrix-vector multiplications [74] are possible applications.

Due to the $\mathcal{NP}$-hardness of hypergraph partitioning [21, 24], it is necessary to use heuristics to keep up with growing instances from a growing set of applications. While there are new distributed and parallel approaches for partitioning problems [36, 64], this work focuses on the *multi-level paradigm* in hypergraph partitioning which is still one of the most important heuristics in that field. It is three-layered consisting of the *coarsening*, *initial partitioning*, and *refinement phase*. In the coarsening phase, the original hypergraph is approximated by gradually smaller ones maintaining the overall structure of it. The initial partitioning phase computes a partition on the smallest approximation of the original hypergraph. Finally, the refinement phase projects the initial partition iteratively to the next level finer hypergraph while refining the partition with the aid of local search algorithms in each step.

Moreover, the coarsening phase tries to build structurally similar approximations. To achieve this, highly connected vertices are contracted because they are very likely to end up in the same block of a partition. However, there are many different rating functions discussed in the literature for the local connectivity of two vertices. While the connectivity of two nodes in a simple graph is just the weight of the edge between them, the hypergraph scenario is more difficult because a hyperedge may contain many hypernodes and, more important, two vertices may be connected through more than one hyperedge with different size and weight. There is lots of research on the three phases of the multi-level paradigm (e.g., in Ref. [27, 31, 39, 42, 58, 65, 68]), however, the selection of a proper rating function in the coarsening phase is still an interesting avenue of research.

Currently, many different rating functions are employed in different partitioners. As a consequence, it is not clear which of these functions is best suited for particular hypergraph instances. This work proposes a machine-learning approach that combines those different rating functions and other useful metrics on the hypergraph to compute a likeliness of two adjacent vertices to end up in the same block of a partition. Our goal is to build a coarsening algorithm that performs well on different types of hypergraphs.

1.2. Contribution

The main contribution of this work is the hypergraph pruning algorithm that is discussed in detail in Section 4. Part of the algorithm is a machine-learning model that has been trained on a heterogeneous set of 100 hypergraphs using local features as well as global statistics on the particular hypergraph instance. If the model's prediction is that a pair of adjacent nodes belongs to the same block of a partition in the output, we contract these nodes reducing the input size for the actual hypergraph partitioner. This prediction is accurate between 98% and 99% on independent test data while classifying between 70% and 73% of the input as same block. These results are a necessity to employ the trained machine-learning model as a coarsening step prior to the actual hypergraph partitioning. The proposed algorithm contracts a not inconsiderable amount of vertices, i.e., we contract on average about 24% of all pins among all different hypergraph classes. Although, the prediction making consumes more time than calculating only a single rating function in coarsening and the final partitions are on average slightly worse than the results of a state-of-the-art partitioner. Nevertheless, an analysis of the trained model reveals some interesting insights on the importance of different rating functions used in the hypergraph partitioning community for coarsening.

1.3. Structure of Thesis

The subsequent Section 2 introduces definitions and notations used throughout this thesis. Thereby, we take a look at hypergraphs and hypergraph partitioning. Also, machine-learning approaches used within this work are briefly introduced. Section 3 shortly summarises related work concerning the multi-level paradigm in hypergraph partitioning as well as the usage of machine-learning techniques for search-space pruning on other problems. The idea behind the proposed approach as well as an explanation of the methodology used is given in Section 4. Also, we present solutions to problems which occurred while training the model. Section 5 evaluates the presented approach containing the experimental setup as well as results yielded. The last Section 6 briefly summarises the previous sections as a whole.

2. Preliminaries

This section shortly introduces the main concepts behind hypergraphs in Section 2.1 as well as the two machine-learning concepts used throughout this work in Section 2.2.

2.1. Hypergraphs

The subsequent sections deal with the basic notions of hypergraphs. The definitions provided have been adapted from Ref. [61].

2.1.1. General Definitions

An *undirected* and *weighted* hypergraph $H = (V, E, c, \omega)$ consists of a set of *hypernodes* V and a set of *hyperedges* E, also known as *nets*, as well as *vertex weights* $c\colon V \to \mathbb{R}_{\geq 0}$ and *net weights* $\omega : E \to \mathbb{R}_{>0}$. The size of the hypernode set V is given by $n := |V|$ and the size of the hyperedge set E is defined by $m := |E|$. Each hyperedge $e \in E$ is a subset of hypernodes $e \subseteq V$. A hypernode $v \in V$ is incident to a net $e \in E$ if $v \in e$. Those $v \in e$ are also called *pins*. The number of pins is given by $p := \sum_{e \in E} |e|$ whereby $|e|$ denotes the number of hypernodes $v \in V$ that are incident to e. This cardinality is also known as hyperedge size or number of pins within e. For a given hypernode $v \in V$, the set of neighbours of v is defined by $\Gamma(v) := \{u \mid \exists e \in E\colon \{v, u\} \subseteq e\}$. Furthermore, $I(v) := \{e \in E \mid v \in e\}$ is the set of all nets that are incident to v. The *degree* of a hypernode v can then be expressed by $deg(v) := |\, I(v)\, |$. For convenience, the two weight functions c and ω may also be extended to sets in the following way: $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. Moreover, two nets e_1 and e_2 are called *parallel* if they contain the same pins, i.e., $e_1 = e_2$.

2.1.2. Partitions and Partitioning Problem

A *k-way partition* Π of a hypergraph $H = (V, E, c, \omega)$ is a set of k blocks V_i, i.e., $\Pi = \{V_1, ..., V_k\}$. In addition to that, $\bigcup_{i=1}^{k} V_i = V$; $V_i \neq \emptyset$ for $1 \leq i \leq k$; and $V_i \cap V_j = \emptyset$ for any $i \neq j$ must hold so that Π is a valid partition. Furthermore, a *k-way partition* Π is called *ε-balanced* if all blocks V_i of Π meet a balance constraint. In particular, $c\,(V_i) \leq L_{max} := (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$ needs to be fulfilled for all $1 \leq i \leq k$. For a given $1 \leq i \leq k$, V_i is called *underloaded* if $c\,(V_i) < L_{max}$ and *overloaded* if $c\,(V_i) > L_{max}$. Given a net $e \in E$, the *connectivity set* $\Lambda(e)$ is defined by $\Lambda(e) := \{V_i \in \Pi \mid V_i \cap e \neq \emptyset\}$. Additionally, the *connectivity* $\lambda(e)$ of a particular net e can be expressed by $\lambda(e) = |\, \Lambda(e)\, |$. Those e for whom $\lambda(e) > 1$ is fulfilled are called *cut-nets* whereas nets with $\lambda(e) = 1$ are known as *internal nets*.

The *k-way hypergraph partitioning problem* entails finding an ε-balanced, k-way partition Π of the hypergraph $H = (V, E, c, \omega)$ while minimising an *objective function* $\mathfrak{f}(\Pi)$. Common objective functions are the *cut-net metric* $\mathfrak{f}_c(\Pi) := \sum_{e \in E'} \omega(e)$ as well as the *connectivity metric* $\mathfrak{f}_\lambda(\Pi) := \sum_{e \in E'} (\lambda(e) - 1)\, \omega(e)$ which will mainly be used within this work. The set E' denotes the set of cut-nets within the given partition Π. Unfortunately, optimising each of these metrics is $\mathcal{NP}$-hard. Compare for example Ref. [61] for more details on that.

Rather than working on a hypergraph $H = (V, E, c, \omega)$ as a whole, it might be useful to *contract* vertices reducing the input size of a partitioning algorithm. *Contracting* a tuple of hypernodes (u, v) with $u, v \in e$, $u \neq v$ for an $e \in E$ means merging v into u. In order to do so, the node weight of u is updated, i.e., $c(u) := c(u) + c(v)$. Also, it is necessary to connect u with the neighbourhood of v by replacing v with u in all nets $e \in I(v) \setminus I(u)$ and removing v in all $e \in I(v) \cap I(u)$. Additionally, parallel edges arisen from this operation are removed except for one. The net weight of the remaining edge will be set to the sum of the removed edges plus the original weight of this edge. Moreover, single vertex nets that have been created by contractions are discarded. *Uncontracting* vertex u reverts the operations within the contraction. Uncontracted vertices are part of the same partition block and the node weight of u is updated to $c(u) := c(u) - c(v)$.

2.2. Machine-Learning

Machine-learning describes a set of algorithms that try to extract knowledge from data through statistical learning. The subsequent section deals with the *logistic regression* model. The definitions provided have been adapted from Ref. [71]. Thereafter, the dimensionality reduction technique PCA is briefly introduced whose definitions also originate from Ref. [71]. Rather than implementing these techniques by hand, we use tools that are introduced later on in this work.

2.2.1. Logistic Regression

Logistic regression, also *logistic classification*, is a statistical method for supervised machine-learning. The use-case within this work requires classifying data in one of two classes ω_1 and ω_2. The goal is to estimate the posterior probabilities $P(\omega_i \,|\, x)$, i.e., the probability that given an input x, x belongs to class ω_i. Naturally, $P(\omega_1 \,|\, x)$ and $P(\omega_2 \,|\, x)$ sum up to 1. For the two class use-case, the regression model is defined as

$$\ln \frac{P(\omega_1 \,|\, x)}{P(\omega_2 \,|\, x)} = \theta_0 + \theta^T x \ , \tag{2.1}$$

whereby the term $\theta_0 + \theta^T x$ with $\theta_0 \in \mathbb{R}, \theta = (\theta_1, ..., \theta_n) \in \mathbb{R}^n$ is also referred to as *logit*. By taking into account that the posteriors sum up to 1 and defining $t := \theta_0 + \theta^T x$, the regression model can be transformed to

$$P(\omega_1 \,|\, x) = \sigma(t)\,, \ \ \sigma(t) := \frac{1}{1 + \exp(-t)} \ , \tag{2.2}$$

whereby $\sigma(t)$ is referred to as the *logistic sigmoid* or *sigmoid link* function. The training samples used to train the *parameter vector* θ and the bias θ_0 are written as (y_n, x_n) with $n \in \{1, ..., N\}$ and $y_n \in \{0, 1\}$. The parameters θ_0 and θ can then be estimated using the likelihood function

$$P(y_1, ..., y_N; \theta_0, \theta) = \prod_{n=1}^{N} \Big(\sigma\Big(\theta_0 + \theta^T x_n\Big)\Big)^{y_n} \Big(1 - \sigma\Big(\theta_0 + \theta^T x_n\Big)\Big)^{1-y_n} \ . \tag{2.3}$$

Using the exponents y_n and $1 - y_n$ is a common way to avoid different cases in the formula, i.e., if $y_n = 1$, the second factor becomes 1 and if $y_n = 0$, the first factor becomes 1. For the machine-learning model, the *negative log-likelihood* function given by

$$L(\theta_0, \theta) = -\sum_{n=1}^{N} \Big(y_n \ln\Big(\sigma\Big(\theta_0 + \theta^T x_n\Big)\Big) + (1 - y_n) \ln\Big(1 - \sigma\Big(\theta_0 + \theta^T x_n\Big)\Big)\Big) \ , \tag{2.4}$$

is minimised. It is also referred to as *cross-entropy error*. Minimisation is done by iteratively calculating gradients, which are for example used within the *gradient descent method*. Refer to Ref. [71] for more information.

2.2.2. Principal Component Analysis (PCA)

Real-world problems often have a high-dimensional feature space. However, the observed systems or processes are usually based on a smaller number of variables that probably can not directly be observed. These variables are projected into feature space which in turn can be observed. The dimensionality reduction technique used within this work is the *Principal Component Analysis* (PCA). Refer to Ref. [71] for more information.

It is assumed that the given input $x_n \in \mathbb{R}^l$, $n \in \{1, ..., N\}$ is a random vector with distribution $\mathcal{N}(0; 1)$. However, if the input is distributed with $\mathcal{N}(\mu; \sigma^2)$, the input should be normalised with $\frac{x_n - \mu}{\sigma}$. The *principal component analysis* consists of iteratively calculating the axes referred to as *principal components* along which the data has its largest remaining variance. If u_1 denotes the first principal component, the variance of the data projected along u_1 can be expressed by

$$\begin{aligned} J(u_1) &= \frac{1}{N} \sum_{n=1}^{N} \left(u_1^T x_n \right)^2 = \frac{1}{N} \sum_{n=1}^{N} \left(u_1^T x_n \right) \left(x_n^T u_1 \right) = u_1^T \hat{\Sigma} u_1 \ , \\ \text{with} \quad \hat{\Sigma} &:= \frac{1}{N} \sum_{n=1}^{N} x_n x_n^T \ . \end{aligned} \tag{2.5}$$

$\hat{\Sigma}$ is the sample covariance matrix which is symmetric and positive semi-definite. To maximise the variance along the direction of u_1, the constrained optimisation problem given by

$$u_1 = \arg\max_u u^T \hat{\Sigma} u \ , \text{so that} \ u^T u = 1 \tag{2.6}$$

is considered. It can be solved using the Lagrangian multiplier

$$L(u, \lambda) = u^T \hat{\Sigma} u - \lambda \left(u^T u - 1 \right) \ . \tag{2.7}$$

Setting its gradient equal to zero yields

$$\hat{\Sigma} u = \lambda u \ . \tag{2.8}$$

In other words, finding the direction along which the sample data has its largest variance is equivalent to finding the normalised eigenvector u to the largest eigenvalue λ. Repeating this for the second largest eigenvalue and so on yields l principal components. Because $\hat{\Sigma}$ is symmetric and positive semi-definite as mentioned before, $\lambda_1 > ... > \lambda_l > 0$. It is very likely, that the first $m < l$ principal components already explain a large amount of the sample variance. If this is the case, the sample vectors $x_n \in \mathbb{R}^l$ might be transformed to $\mathbb{R}^m$ by calculating $(u_1 \,|\, ... \,|\, u_m)^T x_n$ without losing much information.

3. Related Work

This section briefly summarises other works that are related to the presented approach. There is lots of related work on graph partitioning which is summarised in Ref. [9, 66], however, since we focus on hypergraph partitioning, only the most important results will be mentioned in the subsequent sections. First, the *multi-level paradigm* will be introduced in Section 3.1 by explaining the different phases within it. Second, the usage of machine-learning techniques for search-space pruning on other problems will be presented in Section 3.2.

3.1. Multi-level Hypergraph Partitioning

The *multi-level paradigm* has become one of the most important heuristics in hypergraph partitioning. Rather than partitioning a hypergraph directly, the heuristic relies on a three-phase approach which is for example described in Ref. [2, 39, 42]. Fig. 1 illustrates these phases. During the *coarsening phase* a hierarchy of gradually smaller hypergraph approximations is built that try to maintain the overall structure of the hypergraph. The *initial partitioning phase* partitions the smallest approximation of the original hypergraph. Finally, the *uncoarsening phase* tries to successively go from smaller to larger hypergraphs within the hierarchy built by the coarsening phase while performing a local search algorithm for each uncontraction to refine the yielded solutions. This step is also known as *refinement*.

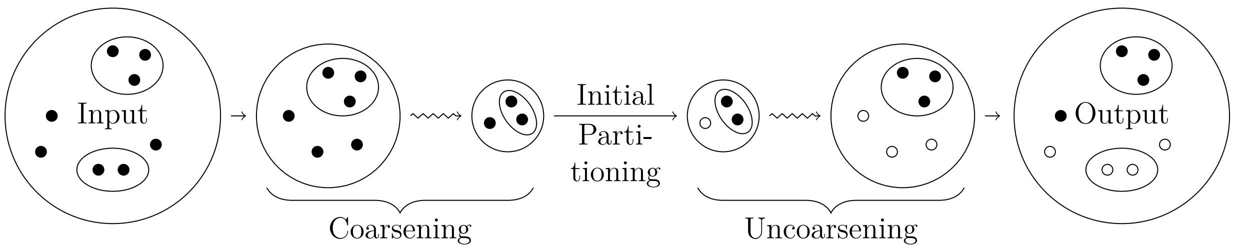


Figure 1: Schematic depiction of the multi-level hypergraph paradigm. Nodes with same colour belong to the same block of a partition.

Moreover, the coarsening phase tries to gradually build approximations that are structurally similar. As already mentioned in Section 1.1, highly connected vertices are contracted because they are very likely to end up in the same block of a partition. However, there are plenty of rating functions that measure the connectivity between two vertices. Some of these connectivity metrics have been used as features in the presented approach. Refer to Section 4.2 for more information.

3.1.1. Coarsening Phase

In the following, coarsening phases of different hypergraph partitioning algorithms are outlined. All these algorithms have in common that they introduce techniques used later on in this work.

dKLFM. The two-level algorithm dKLFM proposed by Goldberg and Burstein [27] is based upon the results yielded by their evaluation of the Fiduccia-Mattheyses algorithm (FM). The

FM algorithm [22] is a greedy algorithm that successively swaps nodes between blocks, one at a time, to iteratively improve the overall solution quality and is still the basis for many bipartioning algorithms. It is also used in a majority of modern hypergraph partitioners up to today [12]. In particular, they analysed the quality of the solutions yielded by the FM algorithm on hypergraphs with different network ratios $r(H)$. The *network ratio* of a hypergraph is defined by $r(H) := (p - m)/n$ whereby p is the number of pins, m the number of nets and n the number of hypernodes. It is a common measure for the denseness of a hypergraph H. Moreover, Goldberg and Burstein [27] found out that on the one hand for hypergraphs with $r(H) < 3$ the FM approach performs poorly whereas for $r(H) > 5$ the results were nearly optimal. Because important hypergraph classes like the hypergraphs produced from VLSI circuit designs have network ratios below 3 (i.e., $1.9 < r(H) < 2.5$ [27]), it is necessary to extend the original FM algorithm. As mentioned earlier, the dKLFM algorithm is a two-level approach. In a first step, a matching is computed and contracted to create a more dense hypergraph regarding to the network ration, i.e., decreasing $r(H)$. Thereafter, a random bipartition of the coarsened hypergraph is used to compute a partition. The FM algorithm is then used to refine the initial partition resulting in a partition for the original hypergraph. The network ratio metric as well as the idea of creating successively more dense approximations of the original hypergraph is used within this work.

HGCEP. The *hierarchical gradual constraint enforcing algorithm* (HGCEP) proposed by Shin and Kim [68] makes use of a clustering technique based on the *closeness* of a pair of vertices to coarsen the hypergraph. In particular, the *closeness* of a pair of hypernodes u, v is defined by

$$\text{closeness}(u, v) := \frac{|\, I(u) \cap I(v) \,|}{\min(\deg(u)\,,\, \deg(v))} - \alpha \cdot \frac{c(u) + c(v)}{\bar{c}} \quad . \tag{3.1}$$

However, successively contracting the nodes that are closest together may produce vertex weights that differ significantly among each other. To deal with this, a weaker form of the balance constraint is initially used. More balanced block weights are then produced by later iterations of the approach which is also eponymous for the algorithm, i.e., *gradual constraint enforcing algorithm*. Since successively contracting nodes maximising a particular rating function is a meta-heuristic often used in coarsening, this idea as well as the employed closeness metric by the HGCEP algorithm is further used throughout this work.

Strawman. The *Strawman* algorithm is a multi-level approach proposed by Hauck [31] and is backed by extensive evaluation of Hauck and Borriello [29, 30] concerning bipartition techniques for the coarsening phase. The resulting algorithm combines several clustering techniques. Besides a random clustering technique based on Ref. [50] and the K-L clustering algorithm [23], the bandwidth clustering approach introduced by Roy and Sechen [58] as well as a newly introduced connectivity clustering algorithm which is inspired by the work of Schuler and Ulrich [65] is used. The bandwidth clustering approach mainly consists of applying its rating function defined by

$$\psi(u, v) := \sum_{e \in I(u) \cap I(v)} \frac{1}{|e| - 1} \quad , \tag{3.2}$$

and contracting hypernodes u with their highest rated neighbours $v \in \Gamma(u)$. The bandwidth metric is a measure for the count of common small nets. The more small common nets there are, the higher are the chances that parallel edges are created when contracting v into u. Moreover, the introduced connectivity clustering algorithm takes this idea even further. The connectivity metric that is used therein can be expressed using the previously introduced bandwidth metric,

$$\mathrm{con}(u,v) := \frac{1}{c(u) \cdot c(v)} \frac{\Psi(u,v)}{(\deg(u) - \Psi(u,v))\,(\deg(v) - \Psi(u,v))} \ . \tag{3.3}$$

Recall that the numerator is a measure for the count of common small nets. Additionally, the denominator ensures that the formed clusters are connected with few other incident nets leading to the formation of clusters that are highly connected in itself but loosely coupled to other clusters. Because of these useful properties, both metrics are used later on in this work. The connectivity clustering approach discussed visits all nodes $u \in V$ of a hypergraph $H = (V, E, c, \omega)$ in random order and contracts node $v \in \Gamma(u)$ with the highest connectivity $\mathrm{con}(u,v)$ into u. This approach also has been adapted by this work but rather than using a fixed rating function, a machine-learning based approach is used to make this kind of decisions.

hMETIS. The hMETIS algorithm is a multi-level partitioning system introduced by Karypis et al. [37, 38, 39]. While the initially proposed version is based on recursive bisection, a newer version uses direct k-way partitioning [41, 42, 43]. The coarsening scheme of the initial version is mainly based on two observations. On the one hand, the coarsening should create approximations on which the initial partitioning algorithm produces similar solution qualities compared to the final partition. On the other hand, a reduction of the pin size p resulting in smaller hyperedges leads to better performances of refinement algorithms especially if the refinement algorithm uses a move-based approach because they tend to work better on smaller nets. The edge coarsening algorithm (EC) that is employed in the hMETIS algorithm visits nodes u in random order similar to the *Strawman* algorithm. In addition to the contraction of node u with node $v \in \Gamma(u)$ maximising a certain rating metric, only *unmatched* nodes are taken into account, so that the vertex weights of the coarse hypergraph are distributed more equally. To be more precise, hypernode u is matched with its unmatched neighbour $v \in \Gamma(u)$ maximising

$$\mathrm{con}(u,v) := \sum_{e \in I(u) \cap I(v)} \frac{\omega(e)}{|e| - 1} \ . \tag{3.4}$$

For unit net weights the connectivity metric of the hMETIS algorithm is equal to the bandwidth clustering rating function in Equation 3.2. However, there are two problems with the matching based coarsening algorithm EC. First, a node u may only be part of one contraction leading to poor progress of the coarsening which in turn requires more iterations of the algorithm. Second, only nets of size 2 can be removed and only if the two pins are matched. There are modifications to the original EC algorithm which are namely the hyperedge coarsening algorithm (HEC) as well as the modified hyperedge coarsening algorithm (MHEC). Both deal with this problem by performing a preprocessing step before the actual coarsening takes place. For more information on that refer to the given references. The approach presented within this work also uses a matching-based rating strategy to coarsen the hypergraph.

KaHyPar. The **Ka**rlsruhe **Hy**pergraph **Par**titioner (KaHyPar) [2, 62] is a direct k-way multi-level partitioner developed at the Karlsruhe Institute of Technology. While other partitioners often use clustering or matching based approaches in coarsening which leads to an approximation hierarchy of $\mathcal{O}(\log n)$ levels, KaHyPar only removes a single hypernode per level yielding $\mathcal{O}(n)$ levels in the hierarchy. This approach is beneficial for the local search heuristic employed in the refinement phase. Moreover, the coarsening phase also takes recourse to the rating function used by hMETIS [39] which is defined in Equation 3.4 and also used by other popular partitioners like Parkway [72] and PaToH [10]. Similar to the coarsening scheme of the Strawman algorithm [31], all nodes are visited in random order and contracted with their neighbour of highest rating according to the employed rating function. This is repeated in several passes until a proper hypergraph size is reached that can be used in the initial partitioning phase or there are no viable contractions left. While other partitioners that use clustering or matching based approaches create a *new* hypergraph at each pass based on the information provided by the matching or clustering, the single contractions made in the KaHyPar algorithm immediately alter the underlying data structure increasing the overall performance because no large-scale hypergraph restructuring is needed. As an initial partitioning algorithm, KaHyPar uses an n-level recursive bisection algorithm together with a pool of other greedy heuristics [62]. The algorithm employed in the refinement phase is based on a local search heuristic inspired by the original FM heuristic [59].

KaHyPar-CA. Heuer and Schlag [35] proposed an addition – KaHyPar-CA – to the original KaHyPar algorithm regarding the employed coarsening. Continuing the thoughts of Karypis and Kumar [41, 42, 43] that coarsening should reduce hyperedge sizes, coarse hypergraphs should contain less nets, and approximations of the original hypergraph should be structurally similar, they point out an approach that identifies community structures in coarsening. A community is a cluster of nodes that are highly connected to each other but rather sparsely connected to other communities. The approach presented is two-fold. First, a community detection algorithm which provides a set $C = \{C_1, ..., C_x\}$ of communities is performed. Thereby, they use the modularity function of Newmann and Girvan [51] to evaluate the quality of the division into communities (disjoint subgraphs). In a second step, a coarsening algorithm is executed on each of these disjoint communities while avoiding the contraction of hypernodes u, v from different communities, i.e., u, v must be in the same community C_i. Only contracting nodes of the same community maintains the overall structure of the original hypergraph in the approximation.

KaHyPar-E. Different from the approaches described before, the KaHyPar-E algorithm [5, 53] is the first multi-level memetic approach on hypergraph partitioning. *Memetic* or *evolutionary* algorithms are inspired by the Darwinian concept of survival of the fittest in evolutionary biology and consist of two main operations, i.e., recombination and mutation. A partitioning algorithm that includes some kind of random selection in the coarsening phase is used to build an initial population of solutions. The *fitness* of an individual may be evaluated by the connectivity objective function $\mathfrak{f}_\lambda$. Thereafter, the initial population is iteratively evolved by recombining and mutating individuals by a given probability using the *steady-state* paradigm [15].

3.1.2. Refinement Phase

Since this work mainly focuses on the preprocessing and contraction of hypergraphs, the techniques used in refinement are only briefly explained. Local search heuristics are used to refine the initial partition along the hierarchy of approximations built by the coarsening phase. Rather than searching for a global optimum regarding the chosen objective function which is infeasible for relevant hypergraph instances due to their sizes, a local search aims to find a local optimum in the neighbourhood of the nodes to uncontract at each level of the hierarchy. Modern hypergraph partitioners such as in Ref. [2, 4, 6, 11, 16, 35, 36, 39, 42, 62, 72, 74] either use variations of the *Fiduccia-Mattheyses* (FM) [22, 59] or *Kernighan-Lin* (KL) [44, 67] algorithm, or even use simpler heuristics with a *greedy* approach [39, 42]. However, recent refinement algorithms also use network flow-based approaches [28, 34].

3.2. Learning Heuristics for Search-Space Pruning

The approach using a machine-learning algorithm to prune search-space of optimisation problems is not new. This section briefly presents two recent approaches using this technique. The approach presented within this work is quite similar to the approaches shown. However, feature selection and related problems are quite different because of different problem domains. Although this difference, the problem of hypergraph partitioning and the optimisation problems that are dealt with in the following approaches coincide in the fact that they are all $\mathcal{NP}$-hard which means that there is not any polynomial-time algorithm for these problems unless $\mathcal{P} = \mathcal{NP}$.

3.2.1. Search-Space Pruning for Clique Detection

Lauri et al. [47] successfully applied logistic classification to prune search-space for clique detection in graphs $G = (V, E)$. They used machine-learning algorithms to prune search-space for the detection algorithm rather than learning the output function of the optimisation problem directly. This difference is illustrated in Fig. 2. The instances that are coloured the

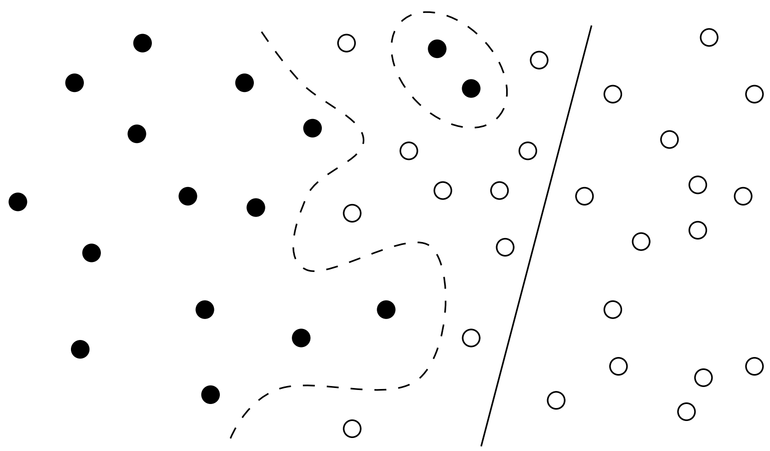


Figure 2: Comparison between pruning search-space and learning exact decisions.

same belong to the same output class, e.g., black nodes belong to class ω_1 (e.g., node belongs to a clique) and white nodes to ω_2 (e.g., node does not belong to a clique). Rather than learning a classification directly for a given input vector x, i.e., $P(\omega_1 \,|\, x) \geq \frac{1}{2} \Leftrightarrow x$ belongs to ω_1 (which is depicted through the dashed curve), they only make a statement about one direction, i.e., $P(\omega_2 \,|\, x) \geq \frac{1}{2} \Rightarrow x$ belongs to ω_2 which refers to the line in Fig. 2. In other words, they prune nodes that are unlikely to be in a clique improving the performance of the actual clique detection algorithm.

There are two categories of computational features used in the employed machine-learning model. On the one hand, they used features based on the nodes of the graph, i.e., $f : V \to \mathbb{R}^n$. On the other hand, features on edges $e = (u, v) \in E$ have been used, i.e., $f_e : E \to \mathbb{R}^n$. The latter is also useful for the approach presented throughout this work because there are features that can be used for edges in graphs as well as for pairs of adjacent pins in hypergraphs. Those features include statistical features using the *Pearson* χ^2-metric [55, 56] as well as similarity measures which originate from set theory and are frequently used in community detection in graphs [1]. The *Pearson* χ^2-metric is defined as

$$\chi^2 := \sum_{v \in A} \frac{(O_v - E_v)^2}{E_v} \quad , \tag{3.5}$$

whereby $A \subseteq V$. The variables O_v and E_v represent the actually **o**bserved and the **e**xpected value for a particular metric. In this work, the χ^2-metric of hypernode degrees is used. Refer to Section 4.2 for more information. Apart from that, similarity measures similar to the ones presented in Ref. [47] are employed which are namely *Jaccard* indices, *Dice* similarity, and *Cosine* similarity. *Jaccard* indices, which are also called *Intersection over Union* (IOU), are used to compare the similarity of the neighbourhoods of adjacent pins u and v. In general, they are defined as

$$J(A, B) := \frac{|\, A \cap B \,|}{|\, A \cup B \,|} \quad , \tag{3.6}$$

whereby the sets A and B are instantiated with the neighbourhoods $\Gamma(u)$ and $\Gamma(v)$. The *Dice* and *Cosine* similarities are also used to express the similarity of the neighbourhoods of adjacent pins. Their definitions can either be found in Section 4.2 or Appendix A.3.

As mentioned before, Lauri et al. [47] used a *supervised* machine-learning approach based on similarity features. There are also other approaches using *unsupervised* learning like for example restricted *Boltzmann* machines [54] which try to acquire information on the unknown distribution of good solutions as well as *Reinforcement* learning approaches [45, 49] which use architecturally difficult *deep* learning models to make predictions about the problem considered. However, these approaches are very complex by design and therefore hard to analyse on a mathematical level. As a consequence of that, it is also unclear which features of the sample data are being exploited in a trained model. These are the reasons, among others, to prefer a *supervised* approach for this work.

3.2.2. Learning Objective Boundaries for Constraint Optimisation Problems

Spieker and Gotlieb [69] propose a similar approach for *Constraint Optimisation Problems* (COP). A Constraint Optimisation Problem consists of a set of variables $\mathcal{X}$, a set of constraints $\mathcal{C}$ on the variable values as well as an objective function $\mathfrak{f}$ to optimise while still fulfilling all constraints $\mathcal{C}$. These kind of problems are part of many applications like for example traffic optimisation [33], optimisation of resource allocation in construction management [32] or utility maximisation problems in economics [57]. The approach presented [69] uses supervised machine-learning techniques to estimate close boundaries for the variables in set $\mathcal{X}$. The proposed machine-learning model has been trained on both global and per-variable (local) feature values. As global features, the number of variables and constraints are used among other global information. Additionally, there are local features that are computed per variable $x \in \mathcal{X}$. These consist of statistical features on the distribution of all values x was assigned to in the computation of the label-providing algorithm. If $\mathcal{A}_x$ denotes the sequence of values x was assigned to in the label-providing algorithm, local features such as the number of different values in $\mathcal{A}_x$, $\min \mathcal{A}_x$, $\max \mathcal{A}_x$, standard deviation, quartiles, means, etc. can be defined. Spieker and Gotlieb also show that although global features are less descriptive regarding the problem instance – same-sized problems may look inherently different – they provide additional information that improve the overall accuracy of the proposed machine-learning model. For that reason, this work also employs global statistics on a given hypergraph instance as well as local features that are evaluated for pairs of adjacent hypernodes.

4. Machine-Learning based Hypergraph Pruning for Partitioning

The subsequent Section 4.1 contains the main idea of the approach presented in this thesis. Also, the selection process of the hypergraph features as well as a few remarks on the feature computation will be given in Section 4.2 and 4.3. Thereafter, we explain the decisions made concerning the model training in Section 4.4. Finally, we introduce the algorithm for the actual hypergraph pruning in Section 4.5 which combines all the building blocks presented.

4.1. Idea

This section aims to introduce the basic workflow behind the presented approach. Altogether, there are three parts, i.e., the sample generation, the model training and the actual pruning algorithm for hypergraph partitioning. Compare Fig. 3 for a rough overview.

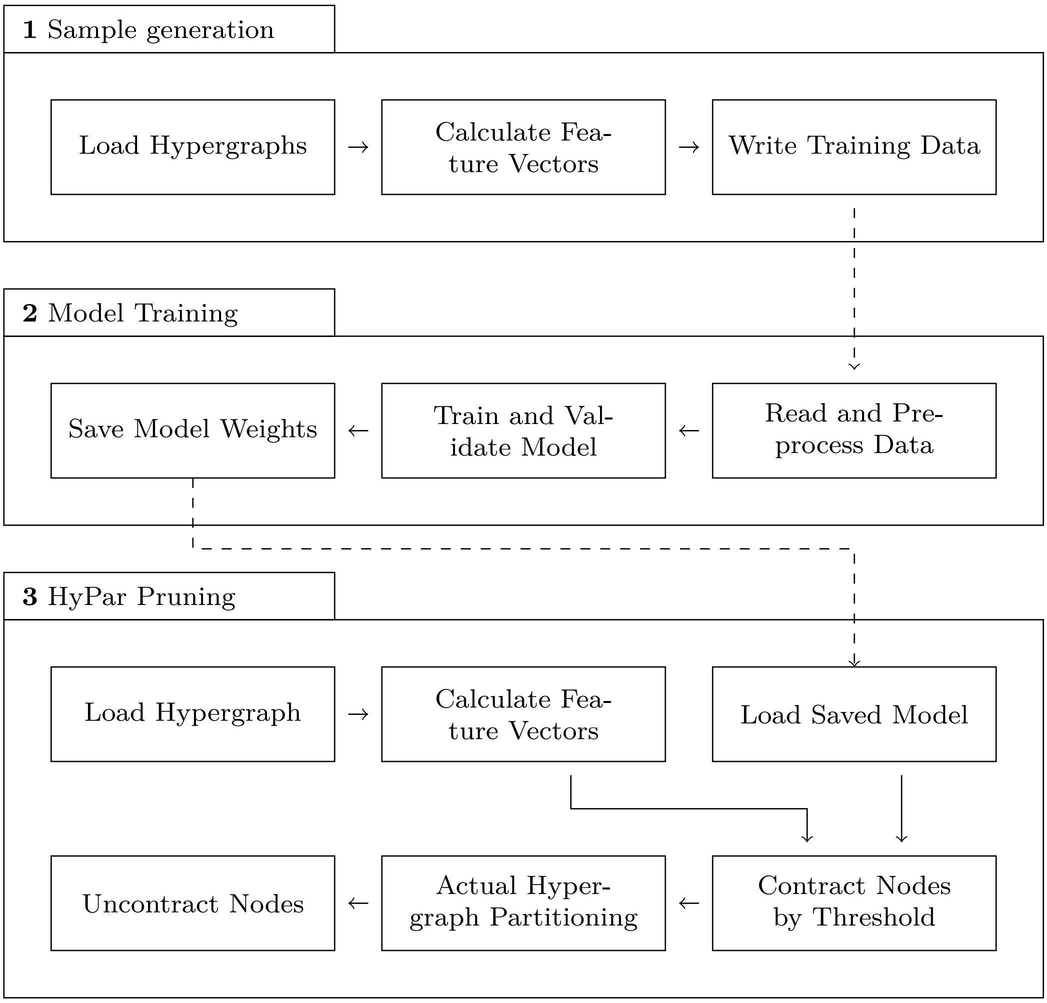


Figure 3: Architecture of the presented approach.

Model training is highly sensitive to quality and amount of the used data. Sample generation loads a predefined set of hypergraphs called the *training set* and calculates feature vectors based on the features that will be presented in the subsequent Section 4.2. Information on the chosen (training) data is given in Section 5.1.1. A feature vector may be calculated for each pair of hypernodes (u, v) with $u \neq v$, $u, v \in e$ for any $e \in E$. Because the number of such pairs increases quadratically with increasing edge size $|e|$, we consider only a linear amount of pairs. Details on the feature computation will be given in Section 4.3. For model training purposes, the feature vectors have to be labelled in order to estimate a pruning function. For a pair of hypernodes (u, v), the class label $y_{u,v} \in \{0, 1\}$ defines whether they belong to the same block of a given partition. Algorithms and configurations used to compute this partition are given in Section 5.1.2.

Model training then uses the previously generated and labelled feature vectors to train a logistic classifier. How to deal with different value ranges and class sizes among the generated features is explained in detail in Section 4.4.

With these two steps done, the actual preprocessing algorithm for hypergraph partitioning is applicable. We use the previously trained model to make predictions about pair of nodes as discussed earlier. Pair of hypernodes that are predicted to be part of the same block in the output are contracted. After applying the actual partitioning algorithm, we uncontract the previously contracted nodes again. More information on this approach is given in Section 4.5.

4.2. Feature Selection

Because of the heterogeneity of the hypergraphs belonging to the training set, we use local features as well as global statistics on the particular hypergraph. Initially, we have considered 37 features that can be divided into the following categories. They are either common hypergraph metrics, features adapted from Ref. [47, 69], or connectivity metrics that are part in the coarsening phase of other partitioning algorithms [27, 31, 58, 65, 68]. In the end, we have selected 25 features by calculating the correlation matrix and iteratively eliminating features that correlate with $\rho > 0.9$. Thereafter, the correlation matrix of the remaining features only contains values less than 0.9. In the following, we only present those 25 selected features. Refer also to Appendix A.3 for a brief overview of all features. Moreover, a *feature vector* $f_{u,v} = (f_1, ..., f_n)^T \in \mathbb{R}^n$ is given by putting the proposed $n = 25$ features into a vector. As mentioned before, we may calculate this vector for each pair of hypernodes (u, v) with $u \neq v$, $u, v \in e$ for any $e \in E$.

4.2.1. Global Hypergraph Features

Besides the standard classification numbers of hypergraphs $H = (V, E, c, \omega)$ which are the number of vertices n (**F1**), the number of edges m (**F2**), and the number of pins p (**F3**), we also consider the *network ratio* (**F4**) as a feature. This ratio is defined by $r(H) := (p - m)/n$ and is a general measure for the overall denseness of a particular hypergraph. In addition to that, the network ratio is quite similar for hypergraph instances originating from the same field of application. Instances that are derived from electrical circuits (VLSI) for example have network ratios in between $1.9 < r(H) < 2.5$ [27] while other classes have other ranges.

Furthermore, statistical features concerning node degrees and edge sizes are used, namely averages, deviations, and quartiles. However, we do not consider the average node degree $\overline{\deg(V)}$ as a feature because of its high correlation with the previously introduced network ratio; compare Ref. [27] or Appendix A.4 for more information. In contrast to that, we consider the standard deviation (**F5**), the minimum (**F6**), the maximum (**F7**) as well as the first quartile of hypernode degrees (**F8**) as features. Median and third quartile of node degrees are dropped because of their high correlation with the network ratio once again. In respect to hyperedge sizes, we use the average (**F9**), the standard deviation (**F10**) as well as the maximum (**F11**) while the minimum and the quartiles of edge sizes are dropped due to high correlation with the minimum (**F6**) and first quartile of hypernode degrees (**F8**) respectively. In total, we use eleven global features to distinguish different hypergraph classes in the regression model applied.

4.2.2. Hypernode Pair Features

Apart from global features, there are also metrics for pairs of adjacent hypernodes (u, v). First, we discuss features working on the neighbourhood of $\Gamma(u)$ and $\Gamma(v)$. Second, we apply statistical measures on this neighbourhood of u and v. Finally, we discuss connectivity measures working with the incident nets $I(u)$ and $I(v)$.

Regarding the neighbourhood $\Gamma(u)$ and $\Gamma(v)$, we use the size of common neighbours $|\,\Gamma(u) \cap \Gamma(v)\,|$ (**F12**), the size of all neighbours $|\,\Gamma(u) \cup \Gamma(v)\,|$ (**F13**) as well as *Jaccard indices* (**F14**) which are defined by

$$J(u, v) := \frac{|\,\Gamma(u) \cap \Gamma(v)\,|}{|\,\Gamma(u) \cup \Gamma(v)\,|} \quad . \tag{4.1}$$

Other neighbourhood similarity features are the *Dice similarity* (**F15**) defined by

$$D(u, v) := \frac{2\,|\,\Gamma(u) \cap \Gamma(v)\,|}{\sum_{w \in \Gamma(u) \cap \Gamma(v)} \deg(w)} \quad , \tag{4.2}$$

which is also known as the F_1-*score* in statistics; as well as the *Cosine similarity* (**F16**) defined by

$$C(u, v) := \frac{|\,\Gamma(u) \cap \Gamma(v)\,|}{\sqrt{\deg(u)\deg(v)}} \quad . \tag{4.3}$$

All these similarity measures have been adapted from Ref. [47].

Regarding the statistical measures of the neighbourhoods, the following features are used. Besides the average of the node degrees of u and v itself (**F17**) and the average of the node degrees of their common neighbours (**F18**), we also use the χ^2-*metric* of hypernode degrees of the common neighbours (**F19**) defined by

$$\chi^2_{deg,\cap}(u, v) := \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{\left(\deg(w) - \overline{\deg(V)}\right)^2}{\overline{\deg(V)}} \quad . \tag{4.4}$$

Analogous to this, we use the average node degrees of all neighbours $\Gamma(u) \cup \Gamma(v)$ (**F20**) as well as the χ^2-*metric* of hypernode degrees of all neighbours (**F21**) defined by

$$\chi^2_{deg,\cup}(u, v) := \sum_{w \in \Gamma(u) \cup \Gamma(v)} \frac{\left(\deg(w) - \overline{\deg(V)}\right)^2}{\overline{\deg(V)}} \quad . \tag{4.5}$$

The usage of χ^2-metrics has been inspired by Lauri et. al. [47].

Finally, we use four connectivity metrics as features that have already proved successful in their application domain. Shin and Kim [68] introduced a *closeness* metric that is used within their HGCEP algorithm and targets the application area of circuits (VLSI). A modified version of the original closeness metric (**F22**) is used as a feature defined by

$$\text{closeness}(u, v) := \frac{|\, I(u) \cap I(v) \,|}{\min(\deg(u)\,, \deg(v))} \quad . \tag{4.6}$$

Moreover, we also use the rating function introduced within the *bandwidth clustering algorithm* of Roy and Sechen [58] (**F23**) as a feature. The rating function is defined by

$$\Psi(u, v) := \sum_{e \in I(u) \cap I(v)} \frac{1}{|\, e| - 1} \tag{4.7}$$

This rating function is also employed in the coarsening phases of the hMetis [39] and the KaHyPar partitioner [2, 62] in a modified version (i.e., edge weights are added to the numerator). To put it bluntly, the bandwidth metric is a measure for the count of common small nets. The more small common nets there are, the higher are the chances that parallel edges are created when contracting v into u. Based on this metric, Schuler and Ulrich [65] propose a *connectivity metric* that incorporates the previously introduced metric. We use a modified version of this connectivity metric (**F24**) defined by

$$\text{connectivity}(u, v) := \frac{\Psi(u, v)}{(\deg(u) - \Psi(u, v))\,(\deg(v) - \Psi(u, v))} \tag{4.8}$$

The original metric is also used within the *Strawman* multi-level algorithm [31]. While the bandwidth metric is a measure for the number of common small nets, the *Strawman* connectivity extends this by taking the neighbourhood of the considered nodes into account. The less nets are incident to u and v apart from the common nets considered, the greater are the values of the metric inducing a *strongly* connected cluster of nodes. Finally, we use the number of common incident nets $|\, I(u) \cap I(v) \,|$ (**F25**) itself as a feature. In total, there are 25 features used throughout this work. Refer to Appendix A.4 for more information about the correlation between them.

4.3. Feature Computation

As mentioned earlier, a feature vector $f_{u,v} = (f_1, ..., f_n)^T \in \mathbb{R}^n$ is given by n ordered feature values regarding pairs of adjacent nodes (u, v). The indices on f are omitted if not necessary in the particular context. To speed up model training, we combine a batch of b feature vectors into a feature matrix $F = \left(f^{(1)} \,|\, ... \,|\, f^{(b)}\right)^T \in \mathbb{R}^{b \times n}$ used to make predictions or train b samples at a time. This matrix is also known as a *training batch*. For implementation details refer to Section 5.1.2.

In order to use a supervised learning approach, we have to provide labels $y_{u,v} \in \{0, 1\}$ for any sample in the set of s samples. To be more precise, $y_{u,v} = 1 \Leftrightarrow u, v \in V_i$ with $i \in \{1, ..., k\}$ for a given k-way partition $\Pi = \{V_1, ..., V_k\}$. This partition Π is calculated by a partitioning algorithm with configuration χ including the number of blocks k and the imbalance parameter ε. The training set $\mathcal{D}_\chi$ can then be expressed as $\mathcal{D}_\chi = \{(f_i, y_i) \mid i \in \{1, ..., s\}\}$. Algorithm 1 calculates this sample set $\mathcal{D}_\chi$.

Algorithm 1: Algorithm for training sample generation

Input: A hypergraph $H = (V, E, c, \omega)$, a hypergraph partitioning algorithm $\text{part}_\chi : H \to \Pi$ with configuration χ containing k, ε and a feature extractor $\text{feature}: V \times V \to \mathbb{R}^n$

$\Pi \leftarrow \text{part}_\chi(H)$ *// Compute partition for labelling*
$\mathcal{D}_\chi \leftarrow \{\}$ *// Set of samples*
foreach $e \in E$ **do**
  Choose $A \subseteq e \times e$, such that $|A| \in \Theta(|e|)$
  foreach $(u, v) \in A$ **do**
    $f_{u,v} \leftarrow \text{feature}(u, v)$ *// Compute feature vector*
    $y_{u,v} \leftarrow \begin{cases} 1 & \text{if } \exists V' \in \Pi : u, v \in V' \\ 0 & \text{else} \end{cases}$ *// label whether nodes belong to same block*
    $\mathcal{D}_\chi \leftarrow \mathcal{D}_\chi \cup \{(f_{u,v}, y_{u,v})\}$

Output: Training sample set $\mathcal{D}_\chi$

First, we use a hypergraph partitioner to obtain a partition Π for labelling purposes as described before. With that completed, we successively compute feature vectors for node pairs (u, v). However, to limit sample size and remove redundant information, we only consider a linear amount of pairs per hyperedge. One possible way of achieving this is by defining $A := \left\{\left(v_i, v_{i+1 \mod |e|}\right) \mid e = \left\{v_1, ..., v_{|e|}\right\}, i \in \{1, ..., |e|\}\right\}$. The set A forming a circle has the benefit of being fully-connected which means that if $G = (V_G, E_G)$ is a simple undirected graph with $V_G = e$ and $E_G = A$, there is a path between each pair of nodes $(a, b) \in V_G \times V_G$. This allows us to maintain information on all (transitive) relations between the pins in the resulting sample set.

4.4. Model Training

This section explains the choices made concerning the training of the machine-learning model. While Section 4.4.1 shows the overall model architecture, the remaining sections deal with details of it. Section 4.4.2 describes the process of input normalisation whereas Section 4.4.3 deals with the reduction of dimensions in feature space. Section 4.4.4 deals with the problems of overfitting and how to overcome them. Thereafter, we describe how to deal with unbalanced class sizes in Section 4.4.5. Finally, we show how the sample data is split for evaluation purposes in Section 4.4.6 as well as how we tune the involved hyperparameters in Section 4.4.7.

4.4.1. Model Architecture

The machine-learning model $\mathcal{M}$ used throughout this work is a logistic regression model with *elastic-net* penalty and *Adam* optimisation. We already have introduced logistic regression in Section 2.2.1, whereas *elastic-net* penalisation is subject to Section 4.4.4. The *Adam* optimisation algorithm [46] can be summarised as an improvement to the traditional gradient descent method that uses moments to avoid getting stuck in local optima. Moreover, the model $\mathcal{M}$ can be expressed as a tuple $\mathcal{M} = (\theta_0, \theta; \beta_1, \beta_2, \lambda, \gamma)$ whereby $\theta = (\theta_1, ..., \theta_n) \in \mathbb{R}^n$ denotes a vector of trainable weights, θ_0 is a trainable variable to model bias in the given data, $\beta_1, \beta_2 \in [0, 1)$ are the hyperparameters for the *Adam optimiser* called the *exponential decay rates* for the moment estimates, and $\lambda \in \mathbb{R}_{>0}, \gamma \in [0,1]$ are *hyperparameters* used for the *elastic-net* penalisation. In the context of machine-learning, a *hyperparameter* is a parameter that is fixed throughout the learning process whereby trainable parameters are iteratively altered in this process.

Training data $\mathcal{D}_\chi$ consists of s samples in the form (f_i, y_i) whereby $f_i \in \mathbb{R}^n$ represents the n-dimensional feature vector and $y_i \in \{0,1\}$ its class label for any $i \in \{1, ..., s\}$. Goal of the model training is to minimise the loss function defined in Equation 2.4. However, this function is further extended due to problems like overfitting, unbalanced classes, and others in the following sections. The final loss function used for model training is given in Equation 4.15. Because the training set $\mathcal{D}_\chi$ depends upon the configuration χ used to compute the partition, we train a separate model $\mathcal{M}_\chi$ for all numbers of blocks k and all imbalance parameters ε for which the model should make predictions.

4.4.2. Input Normalisation

The feature vector f can also be considered as a vector of random variables $F = (F_1, ..., F_n)$ which is useful for the remainder of this section. Due to different data ranges and distributions of the individual feature values, it is hard to train and evaluate a model mainly for the two following reasons that are obtained from Ref. [71]. On the one hand, variables F_i with different expected values $E[F_i]$ are hard to train and to compare since they are not centred, and on the other hand, features with higher variance $V[F_i]$ seem to dominate the model significantly more often, although other features may be more important. To avoid these problems, input is normalised regarding to its distribution. However, Appendix A.5 reveals that the feature values are far from being normally distributed but rather follow a gamma distribution $\Gamma(\alpha, \beta)$ or a log-normal distribution $\text{Lognormal}(\mu, \sigma)$. To choose the best distribution for each individual feature F_i, a *power transform* called *Box-Cox* transformation is used. The transform is given by

$$f_i^{(\lambda_{bc})} = \begin{cases} \dfrac{(f_i+1)^{\lambda_{bc}} - 1}{\lambda_{bc}} & \text{for } \lambda_{bc} > 0 \\ \ln(f_i + 1) & \text{for } \lambda_{bc} = 0 \end{cases} , \tag{4.9}$$

whereby λ_{bc} denotes a parameter used to find the transform with which the data is closest to be normally distributed. This parameter is estimated using a likelihood function. For more details refer to Ref. [8]. Since the *Box-Cox* transformation requires strictly positive variable values but for most of our features only $f_i \geq 0$ holds, we shift all values by one (i.e., $f_i + 1$). Since the transformed features $F_i^{(\lambda_{bc})}$ are close to be normally distributed with $F_i^{(\lambda_{bc})} \sim \mathcal{N}(\mu_i, \sigma_i)$, we can further transform data to zero mean and unit variance by calculating

$$\frac{F_i^{(\lambda_{bc})} - \mu_i}{\sigma_i} \sim \mathcal{N}(0,1) \quad . \tag{4.10}$$

We can also test the transformed distributions for normality by using the *D'Agostino-Pearson* test described in Ref. [13, 52].

4.4.3. Dimensionality Reduction using PCA

As mentioned earlier in Section 2.2.2, systems described by a high-dimensional feature space often rely on a smaller number of not directly observable variables. By reducing the count of variables, we can speed up model training since fewer gradients have to be calculated to find local optima. Appendix A.6 shows that $p = 20$ linear combinations of $n = 25$ feature vector values – the principal components – are enough to explain over 99% of the variance on the training data. A principal component α is given by its coefficients $\alpha = (\alpha_1, ..., \alpha_n)^T \in \mathbb{R}^n$. Principal components are considered in decreasing order of the eigenvalues they refer to, i.e., $\alpha^{(1)}$ is the principal component with the largest eigenvalue and $\alpha^{(n)}$ the principal component with the smallest eigenvalue. Given a feature vector $f \in \mathbb{R}^n$, $\alpha^T f \in \mathbb{R}$ defines a new variable to be used. By using the first $p < n$ principal components and combining them into a matrix $A_p = \left(\alpha^{(1)} \,|\, ... \,|\, \alpha^{(p)}\right)^T \in \mathbb{R}^{p \times n}$, we can reduce the n dimensional feature space to p dimensions by using the feature vector $f^* = A_p f \in \mathbb{R}^p$ in model training instead.

4.4.4. Dealing with Overfitting

Overfitting describes the problem of modelling noise within the process rather than only the process itself. This leads to poor performance on independent test data because of the modelled noise that does not provide any information. To overcome this problem, regularisation is introduced. Regularisation techniques have been adapted from Ref. [20] as well as [71].

The idea of regularisation is to avoid high model complexity by keeping a majority of the trainable weights θ close to zero to avoid modelling noise rather than structural information. This approach is also explained through the *Bias-Variance trade-off* that is described in Ref. [71]. Slightly increasing the bias θ_0 while simultaneously decreasing the variable weights θ yields an overall improvement concerning the *mean-square error*. This behaviour can be accomplished in model training by introducing a penalty term $\Omega(\theta)$ in the loss function

$$L_{reg}(\theta_0, \theta; \lambda) = L(\theta_0, \theta) + \lambda \Omega(\theta) \quad , \tag{4.11}$$

whereby L is the standard logistic regression loss function introduced in Equation 2.4. Common choices for the penalty term are the *least absolute shrinkage and selection operator* LASSO, also known as L_1-*regularisation*, as well as the *ridge* regression, also known as L_2-*regression*, defined by

$$\Omega_{L_1}(\theta) := \|\theta\|_1 = \sum_{i=1}^{n} |\theta_i| \quad \text{and} \quad \Omega_{L_2}(\theta) := \frac{1}{2} \|\theta\|_2^2 = \frac{1}{2} \sum_{i=1}^{n} \theta_i^2 \quad . \tag{4.12}$$

For best results, it is a common practice to use a convex combination of both given by

$$L_{enet}(\theta_0, \theta; \lambda, \gamma) = L(\theta_0, \theta) + \lambda \Omega_{enet}(\theta; \gamma) \quad \text{with} \quad \Omega_{enet}(\theta; \gamma) := \frac{1-\gamma}{2} \sum_{j=1}^{n} \theta_j^2 + \gamma \sum_{j=1}^{n} |\theta_j| \quad . \tag{4.13}$$

This combination is also called *elastic net regularisation*.

4.4.5. Dealing with Unbalanced Class Sizes

There are generally two different ways of dealing with unbalanced class sizes. First, it is possible to reduce the class sizes of classes with too many samples by leaving out some of them; or the other way round, artificially increasing the sample size by duplicating random samples of a particular class adding white random noise to avoid overfitting. These techniques are called under-/oversampling. Second, it is possible to incorporate the class sizes into the model to circumvent too *fast* fitting to the over-represented class. Or to put it in other words, increase the cost of misclassifying samples in the under-represented class.

In this work, we have chosen the second approach for mainly two reasons. On the one hand, leaving out samples antagonises with the goal of a large database of samples. On the other hand, predicting the over-represented class – two nodes belong to the same block of a partition – is more important since it is needed in the pruning algorithm to make these predictions right. However, to avoid overfitting to the over-represented class, cost of misclassifying the under-represented class – i.e., nodes belong not to the same block of a partition ($\omega = 0$) – is increased and cost of misclassifying the other class is decreased. We extend the regularised loss function given in Equation 4.13 by weighting the two classes $\omega \in \{0, 1\}$ differently. If s_0 denotes the count of samples with $y_i = 0$ and s_1 the count of samples with $y_i = 1$ for $i \in \{1, ..., s\}$,

$$c_j = \frac{s_0 + s_1}{s_j} \quad \text{for} \;\; j \in \{0, 1\} \quad , \tag{4.14}$$

defines the cost scaling factors for the two classes. Including these weights yields the final loss function for samples (f_i, y_i), $i \in \{1, ..., s\}$,

$$\begin{aligned} L_{weighted,enet}(\theta_0, \theta; \lambda, \gamma) = -\frac{1}{2s} \sum_{i=1}^{s} \Big(c_1 y_i \ln\Big(\sigma\Big(\theta_0 + \theta^T f_i\Big)\Big) + c_0\,(1 - y_i) \ln\Big(1 - \sigma\Big(\theta_0 + \theta^T f_i\Big)\Big)\Big) \\ + \lambda \left(\frac{1-\gamma}{2} \sum_{j=1}^{n} \theta_j^2 + \gamma \sum_{j=1}^{n} |\theta_j| \right) \quad , \end{aligned} \tag{4.15}$$

which is used by the machine-learning model in this work.

4.4.6. Train-Validation-Test Split

Recall that training data consists of s samples in the form (f_i, y_i) for any $i \in \{1, ..., s\}$. To evaluate a model, we split the set of training samples into a test set and the set used for fitting the model. Each model $\mathcal{M}$ is evaluated against the test set that is kept out from any tuning or training. Again, we split the set used for fitting the model into the actual training set used for estimating θ_0 and θ and the validation set used for tuning the hyperparameters $\beta_1, \beta_2, \lambda, \gamma$. To eliminate bias from the selection of the samples in the training and validation set, we use *k-fold cross validation*. We split the set used to fit the model into k disjoint chunks C_i that resemble the whole set. Thereafter, we train k independent models $\mathcal{M}_i$ with different train-validate splits each for any $i \in \{1, ..., k\}$. Model $\mathcal{M}_i$ uses chunk C_i as validation set and $\bigcup_{j=1}^{k} C_j \setminus C_i$ as training set. Model accuracies are determined by averaging the performance of all those k models. Refer to Ref. [26] for more information about this technique. How the model performances are determined in detail is given in Section 5.1.3.

4.4.7. Tuning Hyperparameters

As already mentioned in Section 4.4.1, our machine-learning model uses several hyperparameters, namely $\beta_1, \beta_2, \lambda, \gamma$. Hyperparameters are fixed in the training process and are used to tune the overall behaviour of fitting the model. To find a tuple $(\beta_1, \beta_2, \lambda, \gamma)$ that performs well enough, we do a *grid-search* using the validation set to tune the hyperparameters. Moreover, the hyperparameters are constrained to a handful of possible values, i.e., $\beta_1 \in \left\{\beta_1^{(1)}, ..., \beta_1^{(a)}\right\}$, $\beta_2 \in \left\{\beta_2^{(1)}, ..., \beta_2^{(b)}\right\}$, $\lambda \in \left\{\lambda^{(1)}, ..., \lambda^{(c)}\right\}$, and $\gamma \in \left\{\gamma^{(1)}, ..., \gamma^{(d)}\right\}$. Because the number of possible combinations is limited, we can train and evaluate all $a \times b \times c \times d$ models in parallel. However, there are heuristics like the *random search* which slightly speed tuning up. For more information about hyperparameter optimisation, refer to Ref. [48].

4.5. Hypergraph Pruning

The meta-algorithm for hypergraph pruning – given in Algorithm 2 – consists of three phases. The first phase contracts node pairs, the second performs the actual partitioning on the contracted hypergraph, and the final phase uncontracts the previously contracted node pairs and performs refinement.

The first phase ranges from line 1 to 16. The goal of the outer loop is to achieve a fixed contraction factor α for any hypergraph instance regarding the number of pins. Optimally, the contracted hypergraph should only contain about $1/\alpha$ of the original number of pins. Moreover, the reason to use the number of pins rather than the number of vertices or edges is that hypergraphs with fewer nodes or edges can still be more difficult to deal with because of higher average net sizes and therefore a higher number of pins. The outer loop repeatedly runs the main part of the contraction algorithm (lines 4-15) until we reach the aimed contraction factor of pins. However, if the contraction algorithm runs out of possible contraction partners – i.e., we contract less than 1% of pins in one pass – the loop exits before accomplishing the number of target pins. Also, we restrict the number of passes to a maximum of 20 passes.

The main part of the contraction algorithm ranges from line 4-15 where we iterate over all unmatched vertices. Moreover, we calculate the prediction values R for each of these nodes u. Thereby, we only use a constant-size and random subset of the neighbours of u. Additionally, we only consider those neighbours that have not been part of a contraction in the respective pass of the outer loop yet. Thereafter, we apply a penalisation function to avoid few heavy nodes. Heavy vertices make it difficult for the initial partitioning to achieve balanced block weights as well as for the refinement phase to move those to other blocks [2, 62]. Refer to Ref. [2, 39, 42] for an overview of best practices in the coarsening phase. Following this, we contract the node u with its neighbour v with which u has the highest likelihood of belonging to the same block. However, this contraction only takes place if this prediction value exceeds a given prediction threshold β. To be more specific, the prediction function calculates the posterior probability $P(y = 1 \mid f_{u,v})$ for a given feature vector $f_{u,v}$. If that probability exceeds the contraction threshold β – i.e., $P(y = 1 \mid f_{u,v}) \geq \beta$ – we merge node v into node u as previously described and remember the pair (u, v) for later uncontraction.

After preprocessing the hypergraph as shown, we use a hypergraph partitioner part_χ to compute a partition for the pruned hypergraph. Because of the rather generic design of the meta-algorithm, the partitioning algorithm part_χ as well as its configuration χ are quite interchangeable which leaves room for optimisation.

We assume that the input hypergraph has unit weights for all nodes and edges since the features introduced in Section 4.2 do not depend on the weight functions c and ω. However, as mentioned in Section 2.1.2, contraction of nodes accumulates their weights introducing non-unit weights. Therefore, the partitioning algorithm part_χ needs to be capable of dealing with weights. Also, it would be possible to add weights to the employed features without changing

Algorithm 2: Meta-algorithm for hypergraph pruning for partitioning

`Input`: A hypergraph $H = (V, E, c, \omega)$ [weights are assumed to be unit weights], a hypergraph partitioning algorithm $\mathrm{part}_\chi \colon H \to \Pi$ with configuration χ, a feature extractor $\mathrm{feature} \colon V \times V \to \mathbb{R}^n$, a prediction function $\mathrm{pred} \colon \mathbb{R}^n \to [0, 1]$, a prediction threshold β, a contraction factor α, a weight penalisation function penalise, a refinement algorithm $\mathrm{refine}(u, v)$, and a maximum count of contraction passes $maxPass$

$P \leftarrow []$ *// Set of contracted nodes*
$pass \leftarrow 0$ *// Current number of iterations*
while $currentNumPins > (1/\alpha)\ initialNumPins$ *and* $pass < maxPass$ **do**
  foreach $u \in V$, u *enabled* and *unmatched* **do**
    Choose $R \subseteq \{\, \mathrm{pred}(\mathrm{feature}(u, v)) \mid v \in \Gamma(u)\,,\ v \text{ unmatched} \,\}$ with $|R| \in \mathcal{O}(1)$ at random
    $R \leftarrow \mathrm{penalise}(R)$ *// Avoid contraction of nodes with high weight by penalising them*
    $v \leftarrow \mathrm{argmax}\, R$ *// Node with which u is most likely to be in the same block*
    $p \leftarrow \max R$ *// Maximum prediction value*
    if v, p *exist and* $p \geq \beta$ **then**
      Contract v into u *// Disables node v and matches u with v in the current pass*
      $P.\mathrm{append}((u, v))$
      if $currentNumPins \leq (1/\alpha)\ initialNumPins$ **then**
        break *// Exit loop if a sufficient amount of pins has already been contracted*
  if *too few progress in the prior step* **then**
    break *// Exit loop if there was too few progress this round*
  $pass \leftarrow pass + 1$
$\Pi \leftarrow \mathrm{part}_\chi(H)$ *// Partition contracted hypergraph;* part *could be a multi-level algorithm itself*
foreach $(u, v) \in P$ *in reversed order* **do**
  Uncontract v from u
  $\Pi(u) \leftarrow \Pi(u) \cup \{v\}$ *// Temporarily add node v to the same block as u*
  $\mathrm{refine}(u, v)$ *// Refine the made uncontraction using a local search heuristic*

`Output`: hypergraph partition Π

the algorithm at all. However, adding support for weights has been left open for future work. Finally, we uncontract the contracted nodes again by initially assigning the block of the representative to the contraction partner. This operation does not violate balance constraints of the overall partitioning since weights are updated as described in Section 2.1.2. Additionally, we apply a refinement algorithm during this last step. Refinement algorithms often use local search heuristics to find local optima in the neighbourhoods of the contraction partners regarding the objective function $\mathfrak{f}$. Details on the specific algorithms used within the presented approach are given in Section 5.1.4.

5. Evaluation

First, we present the experimental setup in Section 5.1 under which the experiments have been conducted. Also, implementation details are given. The second part of this chapter in Section 5.2 contains the results yielded from the experiments done within this work.

5.1. Experimental Setup

This section explains the decisions that have been made for the approach outlined in Section 4.1 on a low level. After taking a look on the used data and the feature computation, implementation details of both the model training and actual pruning algorithm are given.

All implementations in C++ have been compiled using the `gcc` C++ compiler in version 7.5.0 with the `-O3` flag enabled. Moreover, we run all experiments on a machine with 4 Intel® Xeon® Gold 6138 processors with 20 cores each that are clocked at 2 GHz and have 27.5 kiB L1 cache per core, 1 MiB L2 cache per core as well as 27.5 MiB L3 cache shared among all cores. The machine has a total of 754 GiB memory and runs Ubuntu 18.04.4 LTS.

5.1.1. Instances

The used hypergraph data can be divided into two disjoint sets of 100 hypergraph instances each. The training set consists of the hypergraphs given in Appendix A.1, whereas the benchmark set is made up of the hypergraphs given in Appendix A.2. We show an overview of the hypergraph classes in those sets in Table 1. The instances are accessible via the work of Schlag [60]. Schlag has initially collected these instances which form a benchmark set of 488 hypergraphs in total from which the chosen 200 instances are derived. We chose the selected instances by iteratively adding pairs of hypergraphs (one to each set) that have similar node degrees and net sizes. This ensures that both sets contain similar instances regarding size and structure. With more time, however, a more profound analysis could have been done on whether the selected instances are representative of the original set of hypergraphs. Originally,

Class	Training Set	Benchmark Set
DAC2012	5	5
ISPD98	9	9
SAT14 – Primal	21	21
SAT14 – Dual	21	21
SAT14 – Literal	21	21
SPM	23	23
Σ	100	100

Table 1: Number of hypergraph instances per class.

the instances belonging to the DAC2012 class originate from the DAC 2012 Routability-Driven Placement Contest [75], the class ISPD98 consists of hypergraphs from the ISPD98 Circuit Benchmark Suite [3], the SAT14 instances are derived from the SAT competition in 2014 [7] and the SPM class consists of instances from the Sparse Matrix Collection of the University of Florida [14].

We represent boolean satisfiability formulas by interpreting variables of the SAT instance as hyperedges and clauses as hypernodes (primal instances). However, the roles of hyperedges and hypernodes can be swapped in the dual version of the SAT instances. There is also the literal representation where we represent the literals rather than the variables as hypernodes and clauses as hyperedges. Furthermore, we create sparse matrix instances by modelling the dependencies in a matrix vector multiplication [73]. Thereby, rows are interpreted as hypernodes and columns correspond to hyperedges. A non-zero entry in cell (i, j) means that vertex i is part of hyperedge j.

5.1.2. Feature Computation

Partitioner Configuration. As mentioned earlier, the sample set $\mathcal{D}_\chi$ generated by Algorithm 1 and used for model training depends upon the used configuration χ of the partitioner computing the partitions that provide the labels used in the learning process. In the context of this work, χ consists of the parameters k, ε, $\mathfrak{f}$, and C. The parameter k denotes the number of blocks used during partitioning which highly influences the trained model due to different labelling, ε denotes the imbalance parameter, $\mathfrak{f}$ the objective function used and C denotes a set of other parameters that are irrelevant for this work but are needed by the partitioning algorithm employed. All experiments done use the connectivity objective function $\mathfrak{f} = \mathfrak{f}_\lambda$ introduced in Section 2.1.2. Also, we chose $\varepsilon = 0.03$ for all experiments because it is a default value in literature [61]. However, the number of blocks k may vary, i.e., $k \in \{2, 4, 8, 16\}$. On the one hand, we have restricted the amount of possible values of k to four since all steps in Fig. 3 – including sample generation and model training – have to be performed for any additional k. Running all steps for a given number of blocks k took about one to two weeks due to limited resources. On the other hand, the goal of this work is to provide a general contraction algorithm wherefore we used leastwise four different configurations. With the choice of different configurations, it is possible to demonstrate that the accuracy of the predictions is inherent to the chosen machine-learning model and does not depend upon the choice of k. With more time, however, we would have also examined higher numbers of blocks in our experiments, e.g. $k = 64$, $k = 128$ and $k = 256$. An overview of all configurations used is shown in Table 2. The

χ	k	ε	$\mathfrak{f}$	C
χ_2	2	0.03	$\mathfrak{f}_\lambda$	`km1_direct_kway_sea17.ini`
χ_4	4	0.03	$\mathfrak{f}_\lambda$	`km1_direct_kway_sea17.ini`
χ_8	8	0.03	$\mathfrak{f}_\lambda$	`km1_direct_kway_sea17.ini`
χ_{16}	16	0.03	$\mathfrak{f}_\lambda$	`km1_direct_kway_sea17.ini`

Table 2: Configurations used for training sample generation.

partitioner used in the training sample generation algorithm which is part in Algorithm 1 is KaHyPar-CA [35]. KaHyPar-CA outsources its configuration C to configuration files. In particular, we use the parameters given in `km1_direct_kway_sea17.ini`[1] for all experiments.

Implementation Details. Feature computation has been implemented using C++ and the hypergraph datastructures that are part of the KaHyPar project[2]. Those datastructures

[1] https://github.com/kahypar/kahypar/blob/1.2.0/config/km1_direct_kway_sea17.ini
[2] https://kahypar.org/

are also described in Ref. [2, 61]. The feature computation itself is implemented sequentially. However, computing features for different hypergraphs can be run in parallel providing a huge speedup. In total, there are 200 hypergraphs times 4 different configurations resulting in 800 inputs that we have to process.

5.1.3. Model Training

Model Evaluation. The most simple measure for evaluating a model is the accuracy which is defined by the number of right predictions divided by the total number of predictions made. However, this metric provides few insight on performances per prediction class.

Training data once again consists of s samples in the form (f_i, y_i) for any $i \in \{1, ..., s\}$. The class label $\omega \in \{0, 1\}$ predicted by the model for a given f_i is denoted by $\hat{y}_i$. Table 3 contains metrics that are useful for analysing the distributions of classes ω and made predictions. Recall that $\omega = 0$ denotes that two adjacent vertices are not part of the same block whereas $\omega = 1$ means the opposite. Additionally, based on these predictions $\hat{y}_i$, we can define posterior probabilities that are useful for evaluation purposes. Table 4 shows these metrics. Especially

	$\hat{y}_i$	y_i
$\omega = 0$	$P(\hat{y}_i = 0)$	$P(y_i = 0)$
$\omega = 1$	$P(\hat{y}_i = 1)$ (*)	$P(y_i = 1)$

Table 3: Probabilities of classes as well as of the made predictions.

$\hat{y}_i = y_i$	$\hat{y}_i$	y_i
$\omega = 0$	$P(\hat{y}_i = y_i \mid \hat{y}_i = 0)$	$P(\hat{y}_i = y_i \mid y_i = 0)$
$\omega = 1$	$P(\hat{y}_i = y_i \mid \hat{y}_i = 1)$ (*)	$P(\hat{y}_i = y_i \mid y_i = 1)$

Table 4: Posterior probabilities used for evaluation.

the probabilities marked with (*) are important to analyse and to optimise since the model is not used to predict an exact decision boundary but rather to classify a large amount of samples with $\omega = 1$. We have already discussed this thought in Section 3.2. On the one hand, it is important that $P(\hat{y}_i = y_i \mid \hat{y}_i = 1)$ is close to one. Otherwise, the pruning algorithm would withhold node pairs from the actual hypergraph partitioning algorithm that may be part of a cut net. On the other hand, $P(\hat{y}_i = 1)$ should not decrease significantly while optimising the first metric. Otherwise, the pruning may not contract a sufficient amount of nodes. This trade-off between the classification accuracy for $\omega = 1$ and the number of classifications regarding $\omega = 1$ may be tuned by adapting the prediction threshold.

Implementation Details. The presented machine-learning model has been implemented in Python using the machine-learning framework Tensorflow[3]. The logistic regression model as well as the principal component analysis was done within this framework. We have trained four different models on the respective training sample sets, i.e., $\mathcal{D}_{\chi_2}$, $\mathcal{D}_{\chi_4}$, $\mathcal{D}_{\chi_8}$, and $\mathcal{D}_{\chi_{16}}$. Additionally, the trained models have been saved in a binary format provided by the Tensorflow library. The serialised computation graphs also include the preprocessing and scaling of the inputs (i.e., normalising the feature values), so that the application

[3] https://www.tensorflow.org/

of the prediction model does not require additional input scaling or transformations. As described in Section 4.4.1, the model $\mathcal{M}$ consists of several hyperparameters to optimise, i.e., β_1, β_2, λ, γ. For time reasons, the hyperparameters β_1 and β_2 of the employed loss optimisation algorithm have been instantiated with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ without optimising them. Those values are the recommended numbers in the paper presenting the used optimiser [46]. The remaining two parameters have been optimised on the validation sets. See Section 4.4.6 for more information. Thereby, a grid-search on the following value ranges has been done, $\lambda \in \{0.01, 0.001, 0.0001, 0.00001\}$ and $\gamma \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$. Experiments have shown that the maximum accuracy on the respective validation sets has been achieved with $\lambda = 0.0001$ and $\gamma = 0.75$. However, the accuracies only differed by a single-digit percentage. The final accuracies are shown in Section 5.2.1. A more fine-grained tuning of the hyperparameters has been left open for future work since the tuning process is rather time-expensive.

5.1.4. Hypergraph Pruning

Performance Profiles. In order to compare the performances of several algorithms in general, we use *performance profiles* which have been first introduced by Dolan and Moré [17]. All algorithms that are subject to examination are denoted by set $\mathcal{P}$. It is possible that the same algorithm is part of the set multiple times but with different configurations χ. Moreover, $\mathcal{I}$ denotes the benchmark instances that are subject to partitioning. In total, there are $|\mathcal{I}|$ such instances. With these sets in mind, it is possible to define *performance ratios* $r_{p,i}$ given by

$$r_{p,i} := \frac{\mathfrak{f}_\lambda(\Pi_{p,i})}{\min\{\mathfrak{f}_\lambda(\Pi_{q,i}) \mid q \in \mathcal{P}\}} \quad , \tag{5.1}$$

for a particular algorithm p and problem instance i. $\Pi_{p,i}$ denotes the partition with which partitioner p comes up with for hypergraph i. If the partitioners are compared regarding the connectivity metric, the outputs of the connectivity objective function $\mathfrak{f}_\lambda(\Pi)$ are used. Else, the objective function might be replaced by the metric of importance. The connectivity and cut-net objective function have been introduced in Section 2.1.2. The performance profile $r_{p,i} \geq 1$ indicates the factor of how much worse the results of partitioner p on instance i are compared to the best solution for instance i by any partitioner. Furthermore, $r_{p,i} = 1$ if algorithm p performs the best on instance i. *Performance profiles* $\rho_p(\tau)$ can then be expressed by

$$\rho_p(\tau) := \frac{|\{i \in \mathcal{I} \mid r_{p,i} \leq \tau\}|}{|\mathcal{I}|} \quad , \tag{5.2}$$

with $\tau \geq 1$. $\rho_p(1)$ is the fraction of instances for which algorithm p produces the best results. Similarly, $\rho_p(2)$ is the fraction of instances for which algorithm p is at most double as worse as the best algorithm for each instance i. Because the algorithms in the comparison yield results for every instance in the benchmark set, it is not necessary to deal with timeouts or infeasibility in the context of performance profiles.

Because performance profiles $\rho_p(\tau)$ are quite right-skewed, we split the performance profile plots given in Section 5.2.3 into three parts along the x-axis. On the one hand, values near to one are of interest because it is useful to know what fraction of the input instances is solved almost perfectly regarding the available solutions. On the other hand, it is useful to know for which value of τ a majority of instances is better than the best solution times τ. To achieve both, we split the x-axis into three parts to provide the best possible information on the relative performances of the algorithms.

Implementation details. The actual preprocessing algorithm described in Algorithm 2 has been implemented in C++ using the Tensorflow C++ Api to load and apply the trained model to make predictions. Similar to the training sample generation, we use the partitioner KaHyPar-CA [35]. As a refinement algorithm, we use the k-way FM algorithm which is also part of KaHyPar. Refer to Ref. [2, 35] for an overview of the employed local search heuristic. Also, we use the configurations given in Table 2 once again in the contraction algorithm. Naturally, we compare the partitions yielded by our pruning approach with the results produced by KaHyPar-CA itself for different values of k. The comparison is done for each configuration shown before. Section 5.2.3 contains the performance profile plots among other statistics for a comparison of the two algorithms.

5.2. Experimental Results

This section presents the results of this work. First, we evaluate and analyse the trained models. Second, we compare the hypergraph pruning approach described before with KaHyPar-CA.

5.2.1. Model Accuracies

As mentioned before, four different models have been trained that use one of the four sample sets each (i.e., $\mathcal{D}_{\chi_2}$, $\mathcal{D}_{\chi_4}$, $\mathcal{D}_{\chi_8}$, and $\mathcal{D}_{\chi_{16}}$). Table 5 shows an overview of all trained model accuracies in column (1) as well as distributions of classes and predictions, and posterior probabilities. As introduced in Section 5.1.3, the probability in column (2) and the posterior in column (3) is subject to optimisation. The provided information also indicates a consistent

Config	Sample	$P(\hat{y}_i = y_i)$ **(1)**	$P(y_i = 1)$	$P(\hat{y}_i = y_i \mid y_i = 1)$	$P(\hat{y}_i = y_i \mid y_i = 0)$	$P(\hat{y}_i = 1)$ **(2)**	$P(\hat{y}_i = y_i \mid \hat{y}_i = 1)$ **(3)**	$P(\hat{y}_i = y_i \mid \hat{y}_i = 0)$
χ_2	Valid.	0.7419	0.9872	0.7408	0.8287	0.7335	0.9970	0.0397
	Test	0.7413	0.9872	0.7401	0.8293	0.7329	0.9970	0.0396
χ_4	Valid.	0.7320	0.9724	0.7291	0.8346	0.7136	0.9936	0.0804
	Test	0.7305	0.9724	0.7275	0.8384	0.7119	0.9937	0.0802
χ_8	Valid.	0.7338	0.9536	0.7300	0.8135	0.7048	0.9877	0.1277
	Test	0.7333	0.9537	0.7294	0.8145	0.7042	0.9878	0.1276
χ_{16}	Valid.	0.7332	0.9300	0.7281	0.8009	0.6911	0.9798	0.1814
	Test	0.7341	0.9299	0.7291	0.8002	0.6920	0.9798	0.1821

Table 5: Accuracies of the trained model on both the validation and test set.

accuracy between 73% and 74% among all configurations used. Optimising this accuracy as well as adding further configurations is left open for future work due to the time-expensive sample generation and model training process.

5.2.2. Model Analysis

This section analyses the trained models which mainly consist of the trained weights θ. Table 6 qualitatively shows how each of the 25 features is involved in the final model. Since all feature spaces have been transformed to zero mean and unit variance, i.e., $\mathcal{N}(0, 1)$, we can directly compare the trained weights with each other. The symbol $++$ represents weights greater than 1.0, $+$ weights between 0.1 and 1.0, $\circ$ weights between -0.1 and 0.1, $-$ weights between -1.0 and -0.1, and $--$ weights less than -1.0. Keep in mind that the class label $\omega = 1$ means that two nodes belong to the same block of a partition and $\omega = 0$ the opposite. If a weight is negative for example, lower values of the respective feature mean that the likelihood of the two nodes to end up in the same block is increased (since the employed sigmoid kernel is a continuous and strictly monotonically increasing function). Overall, the values of the different configurations are quite consistent among each other. The information that is provided in Table 6 is summarised in the following paragraphs.

Features	Configurations			
	χ_2	χ_4	χ_8	χ_{16}
F01	$--$	$--$	$--$	$--$
F02	$-$	$-$	$-$	$-$
F03	$-$	$\circ$	$+$	$+$
F04	$--$	$--$	$--$	$--$
F05	$--$	$--$	$--$	$--$
F06	$--$	$--$	$--$	$--$
F07	$-$	$-$	$--$	$--$
F08	$-$	$-$	$-$	$-$
F09	$-$	$-$	$-$	$--$
F10	$\circ$	$-$	$--$	$--$
F11	$\circ$	$-$	$-$	$-$
F12	$--$	$--$	$--$	$--$
F13	$\circ$	$+$	$+$	$+$
F14	$+$	$+$	$+$	$+$
F15	$-$	$-$	$-$	$-$
F16	$--$	$-$	$\circ$	$+$
F17	$+$	$+$	$++$	$++$
F18	$+$	$+$	$+$	$+$
F19	$+$	$+$	$\circ$	$-$
F20	$\circ$	$+$	$++$	$++$
F21	$+$	$+$	$+$	$+$
F22	$++$	$++$	$++$	$++$
F23	$-$	$\circ$	$+$	$+$
F24	$++$	$++$	$+$	$-$
F25	$--$	$--$	$--$	$--$

Table 6: Qualitative representation of the trained model weights. $++$ represents weights greater than 1.0, $+$ weights between 0.1 and 1.0, $\circ$ weights between -0.1 and 0.1, $-$ weights between -1.0 and -0.1, and $--$ weights less than -1.0.

First, almost all of the global features are negatively weighted (i.e., F01–F02, F04–F11) which is quite intuitive for the following reason. If a hypergraph is larger or denser in respect of almost any global metric considered (e.g., average edge sizes, network ratio, count of hypernodes, ...), the threshold for the local features to indicate that nodes belong to the same block of a partition is increased. This means that the values of the local features – e.g., average hypernode degree of common neighbours – must be higher to indicate the same as in a less dense hypergraph. An exception to this might be the count of pins p (F03). The higher the number of blocks k is, the more positive is the weighting of it in the resulting model. This can be ascribed to the fact that there are more pair of pins that do not end up in the same block of a partition with increasing k. Compare for example the second numerical column in Table 5 (amount of one-labelled samples).

Second, there are either weights that are consistently negative / positive or weights that change with different partition sizes k among the local features. The first of these categories comprises the features F12–F15, F17–F18, F21–F22, and F25 whereas the second category consists of F16, F19–F20, and F23–F24. Because of the large number of features, we only describe one feature per category. Jaccard indices (F14) are consistently positively weighted among different k. If two nodes share a large amount of their neighbourhood, it is also very likely that they belong to the same block of a partition. By contrast, the weighting of the cosine similarity differs with different partition sizes k. For $k = 2$, the cosine similarity is strongly weighted negative whereas for $k = 16$ it is positively weighted. This may be for the fact that the denominator of the cosine similarity – compare for example Equation 4.3 – contains the geometric mean of the considered node degrees which is a measure of central tendency. The larger the number of blocks k is, the larger may also be the average degree of nodes that still belong to different blocks. Therefore, the feature values need to be weighted more strongly for increasing partition sizes.

Config	Most Important Features (+/−)			
χ_2	F22	1.928	F06	−3.780
	F24	1.638	F05	−2.466
	F21	0.895	F01	−2.378
χ_4	F22	2.069	F06	−4.452
	F24	1.182	F05	−2.498
	F17	0.926	F25	−2.434
χ_8	F22	1.920	F06	−4.215
	F20	1.138	F25	−2.514
	F17	1.077	F05	−2.228
χ_{16}	F22	1.777	F06	−3.942
	F20	1.435	F25	−2.573
	F17	1.181	F05	−1.936

Table 7: Most important features that go into the trained models both on the positive and negative side.

Table 7 shows the three features that are weighted the most positive (left column) as well as the three features that are weighted the most negative (right column) for each configuration χ. There are metrics that are present in all different configurations (i.e., F05, F06, and F22) while there are also metrics whose importance changes with a different number of blocks (i.e., F01, F17, F20, F21, F24, and F25). The most expressive global feature is by far the minimum hypernode degree (F06) followed by the standard deviation of hypernode degrees (F05). Both metrics are well suited for distinguishing hypergraph classes and fitting the regression model even better on different instances. While the standard deviation of node degrees ranges from two to three for ISPD98 instances, the SAT14 primal instances have standard deviations above a value of six; compare Appendix A.1 as well as A.2 for further details. On the local features side, the closeness metric of the HGCEP algorithm [68] (F22) is consistently at the top of the importance ranking among all different configurations. Also, the Strawman connectivity metric [31] (F24) seems to be important especially for small partition sizes, i.e., $k = 2$ or

$k = 4$. Moreover, the χ^2 metric of the degrees of the neighbourhood of the considered pair of nodes (F21) has also importance in the decision making process. Because it is a metric of statistical dispersion, it is detached from scaling issues that come with metrics of central tendency. Also, it incorporates both global and local information.

5.2.3. Hypergraph Pruning

In this section, we evaluate the hypergraph pruning algorithm presented in Section 4.5 concerning both solution quality and time. For numerical stability, we have run the experiments with different random seeds to maintain reproducibility and eliminate bias in the selection of random values within the used algorithms. We combine results yielded by different seeds by using the arithmetic mean. However, when aggregating results further (e.g., to determine the average performance among all instances), we use the geometric mean to give each instance a comparable influence.

β	Hypernodes	Pins	Hyperedges	Avg. Improvement relative to KaHyPar-CA
0.0	0.737	0.495	0.304	1.096
0.1	0.734	0.493	0.307	1.095
0.2	0.700	0.482	0.312	1.101
0.3	0.564	0.401	0.252	1.075
0.4	0.421	0.311	0.184	1.055
0.5	0.310	0.244	0.146	1.047
0.6	0.224	0.188	0.127	1.056
0.7	0.155	0.145	0.100	1.058
0.8	0.085	0.095	0.068	1.057
0.9	0.028	0.067	0.047	0.997
1.0	0.000	0.000	0.000	1.003

Table 8: Contracted amount of hypernodes, pins and hyperedges for different prediction thresholds β. The last column shows the geometric mean of the improvement relative to KaHyPar-CA. Due to limited resources and time, the contraction algorithm was only run on configuration χ_8.

Contraction Ratio. Table 8 shows the amounts of contracted hypernodes, pins and hyperedges for different prediction thresholds β. Our approach aims for a contraction ratio of $1/\alpha = 0.5$. However, the contraction may exit before if we contract less than 1% of pins in the last iteration of the contraction algorithm. The last column shows the relative performances to the KaHyPar-CA partitioner regarding the connectivity metric $\mathfrak{f}_\lambda$. A value above one means that it performs worse than the original algorithm whereas a value below one means the opposite. A prediction threshold of $\beta = 1.0$ indicates that no contractions are made by our approach which corresponds to a normal execution of KaHyPar-CA (therefore the relative performance nearly equal to one). In contrast to that, a prediction threshold of $\beta = 0.0$ indicates that no filtering takes places, i.e., we contract each node with its highest rated neighbour even if it is unlikely that those nodes belong to the same block of a partition. The best improvement relative to KaHyPar-CA is achieved with a prediction threshold of

Class	Hypernodes	Pins	Hyperedges	Avg. Improvement relative to KaHyPar-CA
DAC2012	0.126	0.114	0.086	1.098
ISPD98	0.134	0.080	0.088	0.997
SAT14 – Primal	0.184	0.194	0.185	1.111
SAT14 – Dual	0.588	0.425	0.164	1.077
SAT14 – Literal	0.315	0.207	0.167	1.048
SPM	0.316	0.281	0.093	1.014

Table 9: Contracted amount of hypernodes, pins and hyperedges for $\beta = 0.5$. The last column shows the geometric mean of the improvement relative to KaHyPar-CA. The numbers shown are aggregated from all different configurations χ.

$\beta = 0.9$. However, we have selected the threshold $\beta = 0.5$ since the amount of contractions performed for $\beta = 0.9$ is too little. Apart from $\beta = 0.9$ or $\beta = 1.0$, the prediction threshold $\beta = 0.5$ yields the best relative performances while contracting a not inconsiderable amount of pins. This can be ascribed to the fact that the model training also used a decision boundary of 0.5 to fit the predictions to the provided labels from the sample data. Furthermore, a prediction threshold of $\beta = 0.5$ approximately contracts 31% of the hypernodes, 24% of the pins and 15% of the hyperedges in the initial hypergraph on average. To be more precise, Table 9 shows the amount of contractions per hypergraph class with its respective relative performances only for a prediction threshold of $\beta = 0.5$. Our approach is able to slightly improve the average relative performance on ISPD98 instances. Also, the performance on SPM instances is only slightly worse than on KaHyPar-CA. However, SAT14 primal and DAC2012 instance perform poorly in relation to KaHyPar-CA.

Quality. Fig. 4 shows the performance profile plot for all employed configurations χ aggregated. Unfortunately, the presented approach was not able to outperform the original partitioner KaHyPar-CA. However, our approach is only slightly worse. A *wilcoxon signed-rank test* [25] between the results of KaHyPar-CA and our approach yields a p-value of 0.000 157. Because this value is less than 0.05, the difference between the two algorithms is not statistically significant. There are also many possible optimisations that can still be made with which our approach might produce better results (see Section 6.1). The main shortcoming of the presented approach is, to our beliefs, the lack of node and edge weighting in the calculated features, since the first contraction introduces weights in a unit-weighted hypergraph. Refer to Appendix A.8 for performance profile plots for each configuration χ on its own.

Time. Fig. 5 shows two runtime plots comparing the running times of the proposed approach as well as of KaHyPar-CA. Thereby, the left plot shows the total running times for each instance and configuration aggregated. The right plot compares the running times of the partitioning phase in our approach (i.e., execution of KaHyPar-CA on the coarse hypergraph) and the running times of KaHyPar-CA itself. Our approach is much slower because rather than computing a single rating function, we compute 25 features for all neighbours $v \in \Gamma(u)$. Also, the computation of $\Gamma(u) \cap \Gamma(v)$ is very expensive because it requires $\mathcal{O}(\max(|\Gamma(u)|, |\Gamma(v)|))$ time per neighbour. The heavy-edge metric employed in KaHyPar-CA only requires constant time per neighbour. However, if only considering the running times of the actual partitioning phases (right plot), both partitioning phases require roughly the same amount of time. Moreover, Fig. 6 shows that our contractions accelerate almost half of the partitioned instances among all classes but also slow down the other half.

Fig. 7 further compares the running times of KaHyPar-CA and the partitioning phase in our approach for each hypergraph class on its own. On average, partitioning DAC2012 and SAT14 literal instances is faster after our contraction algorithm, SAT14 primal and SPM instances perform roughly similar, and SAT14 dual and ISPD98 instances are slightly slower.

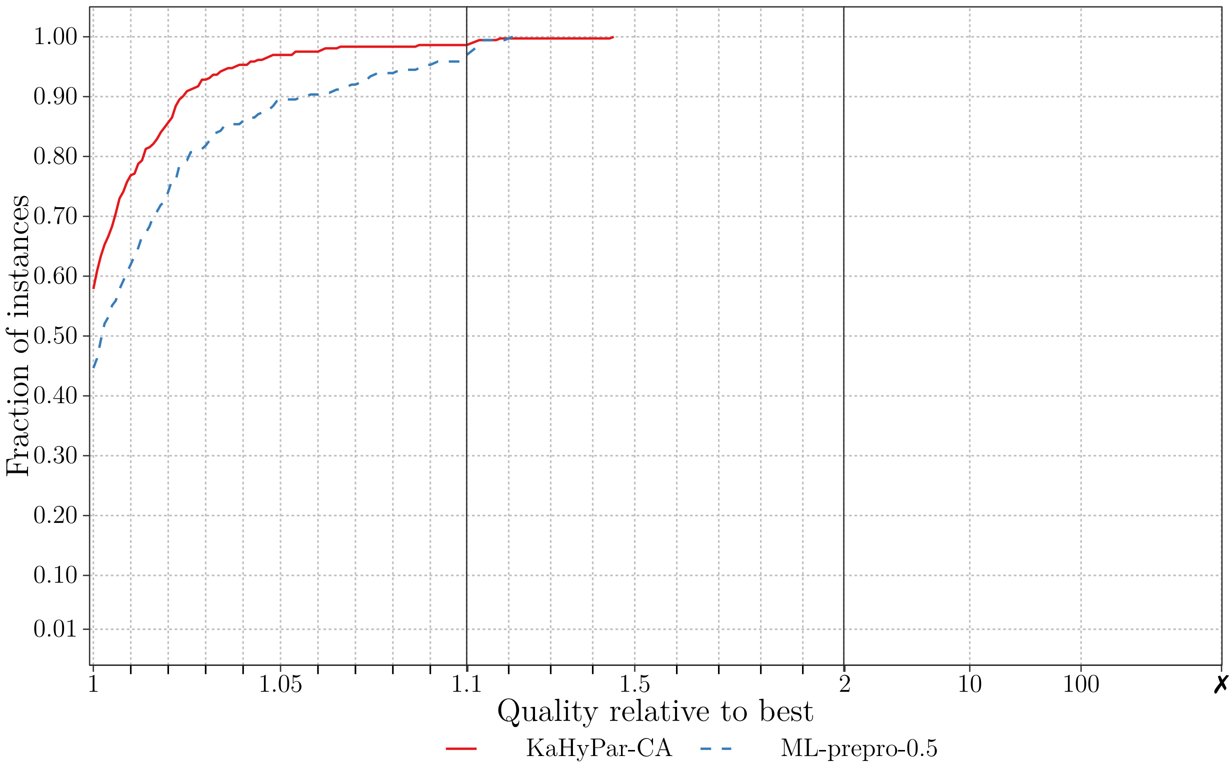
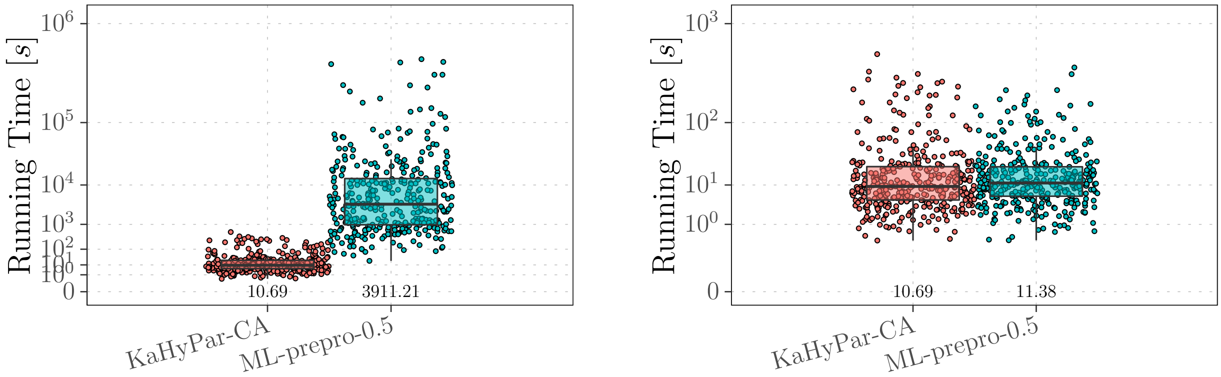


Figure 4: Aggregated performance profile plot for all configurations χ.



(a) Comparison of total running times of KaHyPar-CA and our approach.

(b) Running times of KaHyPar-CA and the partitioning phase in our approach.

Figure 5: Comparison of running times with the KaHyPar-CA partitioner.

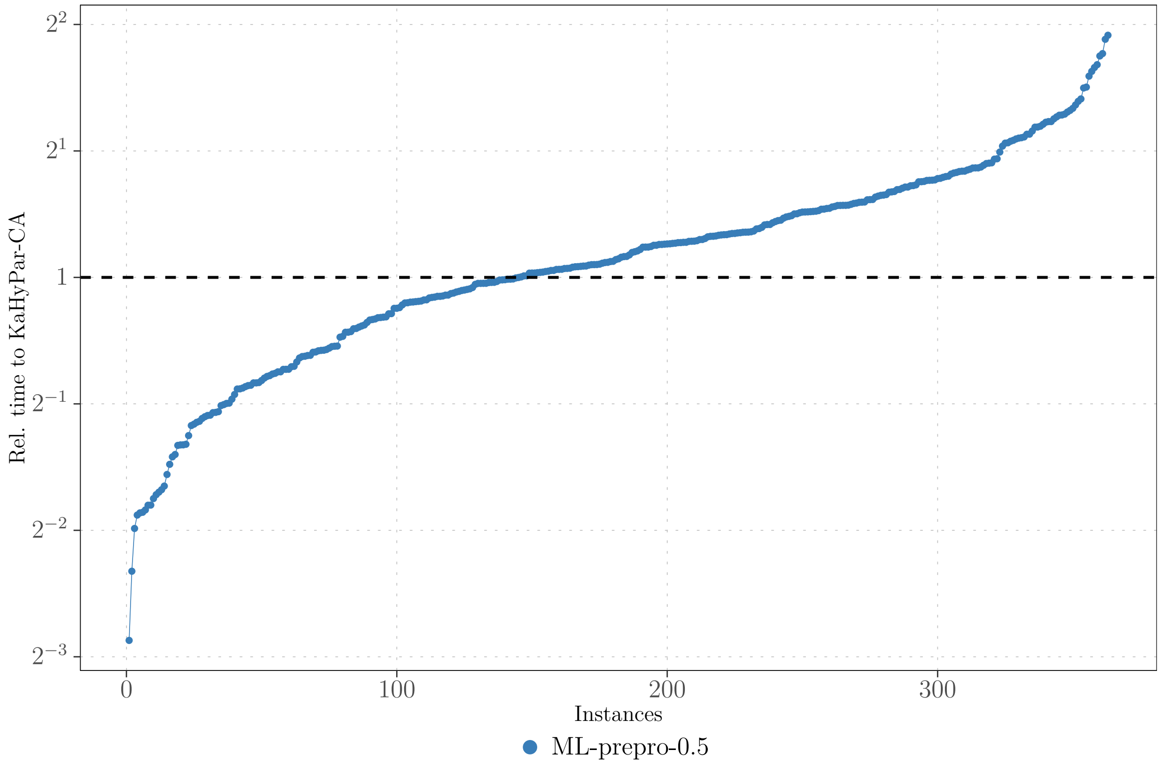
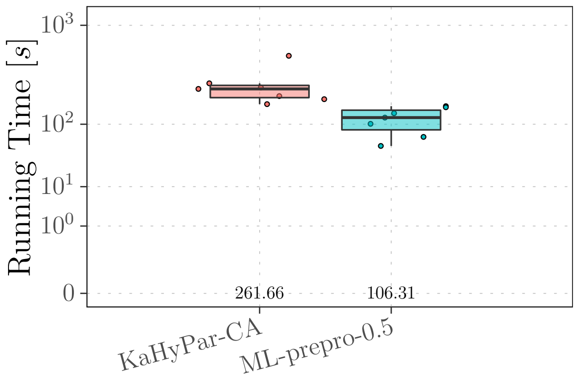
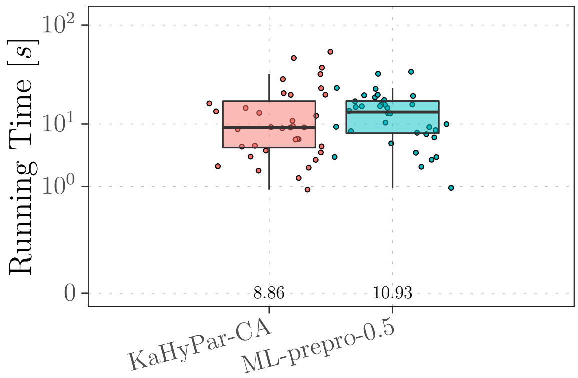


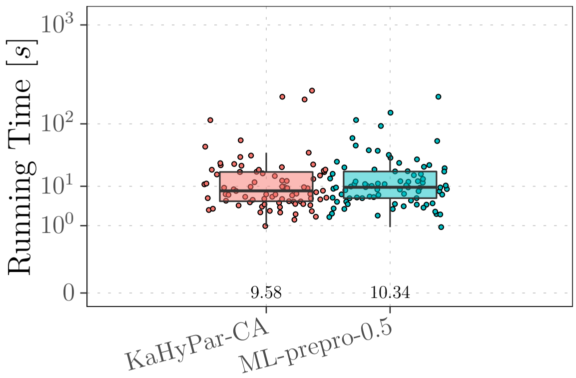
Figure 6: Running times of the partitioning phase in our approach (i.e., execution of KaHyPar-CA on the coarse hypergraph) relative to KaHyPar-CA.



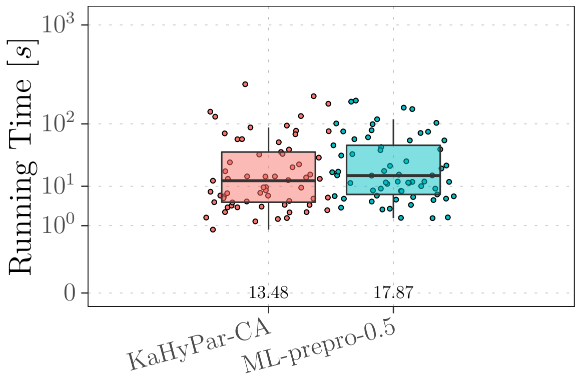
(a) DAC2012 instances.



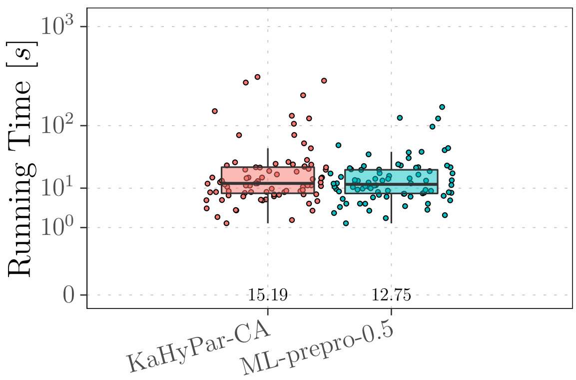
(b) ISPD98 instances.



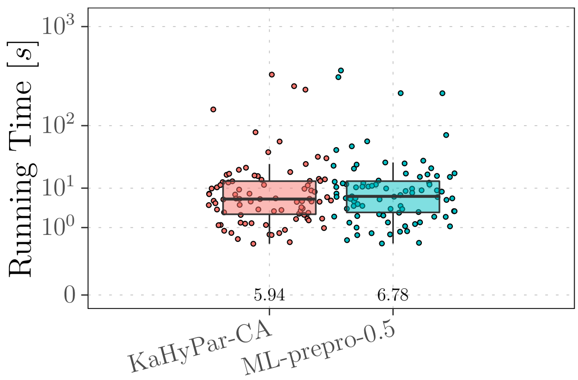
(c) SAT14 primal instances.



(d) SAT14 dual instances.



(e) SAT14 literal instances.



(f) SPM instances.

Figure 7: Comparison of running times of KaHyPar-CA and the partitioning phase in our approach for each hypergraph class.

6. Conclusion

In this work, we have presented a machine-learning based approach with which interesting insights concerning coarsening are connected. The approach consists of a three-fold process. In a first step, we have calculated feature vectors for certain pairs of adjacent nodes. The metrics used in the feature vector are either common hypergraph metrics, statistical measures or coarsening rating functions that are already used by other partitioners (refer back to Section 3). Additionally, we have used a high-quality partitioner to label each feature vector whether a particular pair of nodes belongs to the same block of a partition or not.

In a second step, this information is then used to train a logistic regression model by using advanced techniques such as a principal component analysis or Elastic-Net penalisation for example. Because we transform the feature spaces to zero mean and unit variance normal distributions, we can directly compare the trained weights in order to make statements about their performance. Especially, rating functions such as the closeness metric of the HGCEP algorithm [68] or the Strawman algorithm [31] have been proven to contribute a significant amount to the predictions made.

Finally, we propose a coarsening algorithm that uses the previously trained model to make predictions about the likelihood of belonging to the same block in a partition. On average, the performance of our approach is slightly worse than the performance of KaHyPar-CA. However, this difference is not statistically significant. Regarding running time, the execution of the partitioner on the coarse hypergraphs is on average quite similar to the execution of KaHyPar-CA on the original hypergraphs. We are even able to speed up the partitioning phase on some hypergraph classes. The calculation of the feature vectors, however, makes the approach infeasible. Nevertheless, an analysis of the trained model reveals some interesting insights on the importance of different rating functions used in the hypergraph partitioning community for coarsening.

6.1. Future Work

As already mentioned throughout this work, there are many optimisations possible within the presented approach. Starting with the sample generation process, the labels for the calculated feature vectors have been determined by the partitioner KaHyPar-CA (refer to Section 5.1.2). Although this partitioner is known for producing high quality partitions [2], the model accuracy might be improved by using solutions closer to the optimum. However, partitioners that do so are more time-expensive. Also, the selection of pairs of adjacent hypernodes (u, v) can be optimised to achieve more balanced class sizes while keeping the samples representative in respect of all possible pairs.

Concerning the employed machine-learning model, there are possible optimisations referring to the model architecture as well as the hyperparameters used. We used logistic regression as an underlying model. However, it might be that other approaches produce better accuracies because they fit better to the underlying sample data structure. Possible alternatives that we thought of are *random forests* [71], a *k-nearest neighbours* approach [19] or *support vector machines* [63, 70]. Some unoptimised tests in the beginning of this work, however, lead to the selection of a logistic regression model but nevertheless it might be that other approaches perform better. Moreover, a logistic regression model also has the advantage that weights are easily interpretable in contrast to the numerous *boosted trees* in a random forest approach for example or even more complicated architectures. Within the employed model, there is also room for optimisation. The four hyperparameters used – i.e., β_1, β_2, λ, and γ – can be tuned more fine-grained. As mentioned in Section 4.4.7, a grid-search with more possible values can be done. This endeavour, however, was too time-expensive to be done within this work.

Furthermore, our presented contraction algorithm performs poorly on some of the hypergraph classes whereas it yields acceptable results on other classes. A more detailed analysis of the performance on different hypergraph instances may reveal further insights and shortcomings of our approach. We also thought of reducing the number of features employed to improve the computation time of the feature vector calculations. Apart from that, the main shortcoming of the presented approach is, to our beliefs, the lack of node and edge weighting in the calculated features, since the first contraction introduces weights in a unit-weighted hypergraph. The heavy-edge metric employed in KaHyPar-CA for example incorporates hyperedge weights to make contraction decisions.

References

[1] L. A. Adamic and E. Adar. Friends and Neighbors on the Web. *Soc. Networks*, 25(3):211–230, 2003.

[2] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag. Engineering a direct k-way Hypergraph Partitioning Algorithm. In *Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2017*, pages 28–42. SIAM, 2017.

[3] C. J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proceedings of the 1998 International Symposium on Physical Design, ISPD 1998*, pages 80–85. ACM, 1998.

[4] C. J. Alpert, J. Huang, and A. B. Kahng. Multilevel Circuit Partitioning. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(8):655–667, 1998.

[5] R. Andre, S. Schlag, and C. Schulz. Memetic Multilevel Hypergraph Partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 347–354. ACM, 2018.

[6] C. Aykanat, B. B. Cambazoglu, and B. Ucar. Multi-level direct k-way Hypergraph Partitioning with Multiple Constraints and Fixed Vertices. *J. Parallel Distributed Comput.*, 68(5):609–625, 2008.

[7] A. Belov, D. Diepold, M. Heule, and M. Järvisalo. The SAT Competition 2014. 2014.

[8] G. E. P. Box and D. R. Cox. An Analysis of Transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26:211–252, 1964.

[9] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent Advances in Graph Partitioning. In *Algorithm Engineering – Selected Results and Surveys*, pages 117–158. 2016.

[10] Ü. V. Catalyürek and C. Aykanat. PaToH: Partitioning Tool for Hypergraphs. Technical report, The Ohio State University, 1999.

[11] Ü. V. Catalyürek, M. Deveci, K. Kaya, and B. Ucar. UMPa: A Multi-objective, Multi-level Partitioner for Communication Minimization. In *Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*, pages 53–66. American Mathematical Society, 2012.

[12] J. Cong and J. R. Shinnerl. *Multilevel Optimization in VLSICAD.* Kluwer Academic Publishers, 2003.

[13] R. B. D'Agostino. Transformation to Normality of the Null Distribution of g_1. *Biometrika*, 57:679–681, 1970.

[14] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[15] K. de Jong. Evolutionary Computation: a Unified Approach. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 327–342. ACM, 2020.

[16] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and Ü. V. Catalyürek. Parallel Hypergraph Partitioning for Scientific Computing. In *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*. IEEE, 2006.

[17] E. D. Dolan and J. J. Moré. Benchmarking Optimization Software with Performance Profiles. *Math. Program.*, 91(2):201–213, 2002.

[18] V. Durairaj and P. Kalla. Guiding CNF-SAT Search via Efficient Constraint Partitioning. In *2004 International Conference on Computer-Aided Design, ICCAD 2004*, pages 498–501. IEEE Computer Society / ACM, 2004.

[19] R. Fathi, A. R. Molla, and G. Pandurangan. Efficient Distributed Algorithms for the k-Nearest Neighbors Problem. In *32nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2020*, pages 527–529. ACM, 2020.

[20] A. C. Faul. *A Concise Introduction to Machine Learning.* Machine Learning and Pattern Recognition. Chapman and Hall, 2020.

[21] A. E. Feldmann. Fast Balanced Partitioning is hard even on Grids and Trees. In *Mathematical Foundations of Computer Science 2012 – 37th International Symposium, MFCS 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 372–382. Springer, 2012.

[22] C. M. Fiduccia and R. M. Mattheyses. A Linear-time Heuristic for Improving Network Partitions. In *Proceedings of the 19th Design Automation Conference, DAC 1982*, pages 175–181. ACM/IEEE, 1982.

[23] J. Garbers, H. J. Prömel, and A. Steger. Finding Clusters in VLSI Circuits. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1990*, pages 520–523. IEEE Computer Society, 1990.

[24] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[25] E. A. Gehan. A Generalized Wilcoxon Test for Comparing Arbitrarily Singly-Censored Samples. *Biometrika*, 52(1–2):203–224, June 1965.

[26] B. Ghojogh and M. Crowley. The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial. *CoRR*, abs/1905.12787, 2019.

[27] M. K. Goldberg and M. Burstein. Heuristic Improvement Technique for Bisection of VLSI Networks. In *International Conference on Computer-Aided Design (ICCAD)*, pages 122–125, 1983.

[28] L. Gottesbüren, M. Hamann, S. Schlag, and D. Wagner. Advanced Flow-Based Multilevel Hypergraph Partitioning. In *18th International Symposium on Experimental Algorithms, SEA 2020*, volume 160 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[29] S. Hauck and G. Borriello. An Evaluation of Bipartitioning Techniques. In *16th Conference on Advanced Research in VLSI, ARVLSI 1995*, pages 383–403. IEEE Computer Society, 1995.

[30] S. Hauck and G. Borriello. An Evaluation of Bipartitioning Techniques. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 16(8):849–866, 1997.

[31] S. A. Hauck. *Multi-FPGA Systems.* PhD thesis, 1995.

[32] T. Hegazy. Optimization of Resource Allocation and Leveling Using Genetic Algorithms. *Journal of Construction Engineering and Management*, 125(3):167–175, June 1999.

[33] M. Herty and A. Klar. Modeling, Simulation, and Optimization of Traffic Flow Networks. *SIAM J. Scientific Computing*, 25(3):1066–1087, 2003.

[34] T. Heuer, P. Sanders, and S. Schlag. Network Flow-Based Refinement for Multilevel Hypergraph Partitioning. *ACM J. Exp. Algorithmics*, 24(1):2.3:1–2.3:36, 2019.

[35] T. Heuer and S. Schlag. Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In *16th International Symposium on Experimental*

Algorithms, SEA 2017, volume 75 of *LIPIcs*, pages 21:1–21:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[36] I. Kabiljo, B. Karrer, M. Pundir, S. Pupyrev, A. Shalita, Y. Akhremtsev, and A. Presta. Social Hash Partitioner: A Scalable Distributed Hypergraph Partitioner. *Proc. VLDB Endow.*, 10(11):1418–1429, 2017.

[37] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. In *Proceedings of the 34st Conference on Design Automation, 1997*, pages 526–529. ACM Press, 1997.

[38] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Application in VLSI Domain. Technical report, University of Minnesota, 1997.

[39] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Trans. Very Large Scale Integr. Syst.*, 7(1):69–79, 1999.

[40] G. Karypis, R. Aggarwal, V. Kurnar, and S. Shekhar. Multilevel Hypergraph Partition: Applications in VLSI Design. January 1997.

[41] G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. Technical report, University of Minnesota, 1998.

[42] G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. In *Proceedings of the 36th Conference on Design Automation, 1999*, pages 343–348. ACM Press, 1999.

[43] G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. *VLSI Design*, 2000(3):285–300, 2000.

[44] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Syst. Tech. J.*, 49(2):291–307, 1970.

[45] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning Combinatorial Optimization Algorithms over Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 6348–6358, 2017.

[46] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[47] J. Lauri, S. Dutta, M. Grassia, and D. Ajwani. Learning fine-grained Search Space Pruning and Heuristics for Combinatorial Optimization. *CoRR*, abs/2001.01230, 2020.

[48] P. Liashchynskyi and P. Liashchynskyi. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *CoRR*, abs/1912.06059, 2019.

[49] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. K. Singh. Learning Heuristics over Large Graphs via Deep Reinforcement Learning. *CoRR*, abs/1903.03332, 2019.

[50] T. N.Bui, C. Heigham, C. Jones, and F. T. Leighton. Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms. In *Proceedings of the 26th ACM/IEEE Design Automation Conference, 1989*, pages 775–778. ACM Press, 1989.

[51] M. E. J. Newman and M. Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review E*, February 2004.

[52] E. S. Pearson. Note on Tests for Normality. *Biometrika*, 22:423–424, 1931.

[53] M. Popp, S. Schlag, C. Schulz, and D. Seemaier. Multilevel Acyclic Hypergraph Partitioning. *CoRR*, abs/2002.02962, 2020.

[54] M. Probst, F. Rothlauf, and J. Grahl. Scalability of using Restricted Boltzmann Machines for Combinatorial Optimization. *Eur. J. Oper. Res.*, 256(2):368–383, 2017.

[55] T. R. C. Read and N. A. C. Cressie. Goodness-of-fit Statistics for Discrete Multivariate Data. *Springer Series in Statistics*, 1988.

[56] T. R. C. Read and N. A. C. Cressie. Pearson's χ^2 and the Likelihood Ratio Statistic G^2: a Comparative Review. *International Statistical Review*, 57(1):19–43, 1989.

[57] J. Rotemberg and M. Woodford. An Optimization-based Econometric Framework for the Evaluation of Monetary Policy. *NBER Macroeconomics Annual*, 12:297–346, 1997.

[58] K. Roy and C. Sechen. A Timing Driven N-Way Chip and Multi-Chip Partitioner. In *International Conference on Computer-Aided Design (ICCAD)*, pages 240–247, 1993.

[59] L. A. Sanchis. Multiple-Way Network Partitioning. *IEEE Trans. Computers*, 38(1):62–81, 1989.

[60] S. Schlag. Benchmark Sets used in the Dissertation of Sebastian Schlag. 2019.

[61] S. Schlag. *High-Quality Hypergraph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020.

[62] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. k-way Hypergraph Partitioning via n-Level Recursive Bisection. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016*, pages 53–67. SIAM, 2016.

[63] S. Schlag, M. Schmitt, and C. Schulz. Faster Support Vector Machines. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019*, pages 199–210. SIAM, 2019.

[64] S. Schlag, C. Schulz, D. Seemaier, and D. Strash. Scalable Edge Partitioning. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019*, pages 211–225. SIAM, 2019.

[65] D. M. Schuler and E. G. Ulrich. Clustering and Linear Placement. In *Proceedings of the 9th Design Automation Workshop, DAC 1972*, pages 50–56. ACM, 1972.

[66] C. Schulz and D. Strash. Graph Partitioning: Formulations and Applications to Big Data. In *Encyclopedia of Big Data Technologies*. Springer, 2019.

[67] D. G. Schweikert and B. W. Kernighan. A proper Model for the Partitioning of Electrical Circuits. In *Proceedings of the 9th Design Automation Workshop, DAC 1972*, pages 57–62. ACM, 1972.

[68] H. Shin and C. Kim. A Simple yet Effective Technique for Partitioning. *IEEE Trans. Very Large Scale Integr. Syst.*, 1(3):380–386, 1993.

[69] H. Spieker and A. Gotlieb. Learning Objective Boundaries for Constraint Optimization Problems. *CoRR*, abs/2006.11560, 2020.

[70] A. Tharwat. Behavioral Analysis of Support Vector Machine Classifier with Gaussian Kernel and Imbalanced Data. *CoRR*, abs/2007.05042, 2020.

[71] S. Theodoridis. *Machine learning: a Bayesian and Optimization Perspective*, volume 2. Academic Press, London, 2020.

[72] A. Trifunovic and W. J. Knottenbelt. Parallel Multilevel Algorithms for Hypergraph Partitioning. *J. Parallel Distributed Comput.*, 68(5):563–581, 2008.

[73] B. Ucar and C. Aykanat. Revisiting Hypergraph Models for Sparse Matrix Partitioning. *SIAM Review*, 49(4):595–603, 2007.

[74] B. Vastenhouw and R. H. Bisseling. A Two-Dimensional Data Distribution Method for Parallel Sparse Matrix-Vector Multiplication. *SIAM Review*, 47(1):67–95, 2005.

[75] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li, and Y. Wei. The DAC 2012 Routability-driven Placement Contest and Benchmark Suite. In *The 49th Annual Design Automation Conference 2012, DAC 2012*, pages 774–782. ACM, 2012.

[76] Y. Xu and G. Tan. An Offline Road Network Partitioning Solution in Distributed Transportation Simulation. In *16th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2012*, pages 210–217. IEEE Computer Society, 2012.

[77] W. Yang, G. Wang, Md. Z. A. Bhuiyan, and K. K. R. Choo. Hypergraph Partitioning for Social Networks based on Information Entropy Modularity. *J. Netw. Comput. Appl.*, 86:59–71, 2017.

A. Appendix

This section contains detailed information on the used hypergraphs and benchmarking data as well as information on the model training and feature selection process.

A.1. Hypergraph Training Set

Overview of all hypergraph instances used for sample generation for model training. $\overline{\deg(V)}$ denotes the average hypernode degrees and $\sigma(\deg(V))$ the standard deviations of node degrees. Similarly, $\overline{|e|}$ denotes the average net sizes and $\sigma(|e|)$ the standard deviation of net sizes.

Type	Hypergraph	n	m	p	$\overline{\deg(V)}$	$\sigma(\deg(V))$	$\overline{\lvert e\rvert}$	$\sigma(\lvert e\rvert)$
DAC12	superblue14	630 802	619 815	2 048 903	3.248	4.659	3.306	16.171
	superblue19	522 482	511 685	1 713 796	3.280	5.917	3.349	27.175
	superblue3	917 944	898 001	3 109 446	3.387	5.243	3.463	15.285
	superblue6	1 011 662	1 006 629	3 387 521	3.348	4.062	3.365	14.092
	superblue9	844 332	833 808	2 898 403	3.433	4.748	3.476	23.529
ISPD98	ibm04	27 507	31 970	105 859	3.848	4.654	3.311	2.923
	ibm05	29 347	28 446	126 308	4.304	2.354	4.440	4.291
	ibm06	32 498	34 826	128 182	3.944	1.842	3.681	3.278
	ibm08	51 309	50 513	204 890	3.993	6.180	4.056	5.008
	ibm10	69 429	75 196	297 567	4.286	3.218	3.957	3.560
	ibm12	71 076	77 240	317 760	4.471	4.677	4.114	3.719
	ibm14	147 605	152 772	546 816	3.705	3.182	3.579	2.943
	ibm16	183 484	190 048	778 823	4.245	2.769	4.098	3.614
	ibm18	210 613	201 920	819 697	3.892	1.903	4.060	3.963
Primal	6s133	48 215	140 968	328 924	6.822	16.322	2.333	0.471
	6s153	85 646	245 440	572 692	6.687	11.635	2.333	0.471
	6s184	33 365	97 516	227 536	6.820	16.907	2.333	0.471
	6s9	34 317	100 384	234 228	6.825	16.825	2.333	0.471
	aaai10-planning-ipc5-pathways-17-step21	53 919	308 235	690 466	12.806	10.087	2.240	1.884
	ACG-20-5p0	324 716	1 390 931	3 269 132	10.068	8.767	2.350	0.923
	ACG-20-5p1	331 196	1 416 850	3 333 531	10.065	8.758	2.353	0.917
	AProVE07-27	7729	29 194	77 124	9.979	38.729	2.642	1.429
	atco-enc1-opt2-05-4	14 636	386 163	1 652 800	112.927	248.398	4.280	1.364
	atco-enc1-opt2-10-16	9643	152 744	641 139	66.488	142.597	4.197	1.582
	atco-enc2-opt1-05-21	56 533	526 872	2 097 393	37.100	139.375	3.981	1.538
	atco-enc2-opt1-15-100	58 752	580 963	2 227 755	37.918	134.900	3.835	1.534
	bob12s02	26 294	77 920	181 812	6.915	8.104	2.333	0.471
	countbitssrl032	18 607	55 724	130 020	6.988	10.435	2.333	0.471
	dated-10-11-u	141 860	629 461	1 429 872	10.080	4.955	2.272	0.935
	dated-10-17-u	229 544	1 070 757	2 471 122	10.765	6.790	2.308	0.920
	gss-19-s100	31 435	94 548	222 806	7.088	6.516	2.357	0.480
	hwmcc10-timeframe-expansion-k45-pdtvisns3p02-tseitin	163 622	488 120	1 138 944	6.961	15.164	2.333	0.471
	itox-vc1130	152 256	441 729	1 143 974	7.513	47.981	2.590	0.537
	manol-pipe-c8nidw	269 048	799 867	1 866 355	6.937	16.726	2.333	0.471
	manol-pipe-g10bid-i	266 405	792 175	1 848 407	6.938	21.682	2.333	0.471
Dual	6s133	140 968	48 215	328 924	2.333	0.471	6.822	16.322
	6s153	245 440	85 646	572 692	2.333	0.471	6.687	11.635
	6s184	97 516	33 365	227 536	2.333	0.471	6.820	16.907
	6s9	100 384	34 317	234 228	2.333	0.471	6.825	16.825
	aaai10-planning-ipc5-pathways-17-step21	308 235	53 919	690 466	2.240	1.884	12.806	10.087
	ACG-20-5p0	1 390 931	324 716	3 269 132	2.350	0.923	10.068	8.767
	ACG-20-5p1	1 416 850	331 196	3 333 531	2.353	0.917	10.065	8.758
	AProVE07-27	29 194	7729	77 124	2.642	1.429	9.979	38.729
	atco-enc1-opt2-05-4	386 163	14 636	1 652 800	4.280	1.364	112.927	248.398
	atco-enc1-opt2-10-16	152 744	9643	641 139	4.197	1.582	66.488	142.597
	atco-enc2-opt1-05-21	526 872	56 533	2 097 393	3.981	1.538	37.100	139.375
	atco-enc2-opt1-15-100	580 963	58 752	2 227 755	3.835	1.534	37.918	134.900
	bob12s02	77 920	26 294	181 812	2.333	0.471	6.915	8.104
	countbitssrl032	55 724	18 607	130 020	2.333	0.471	6.988	10.435

Table 10: Overview of hypergraph instances in training set.

Type	Hypergraph	n	m	p	$\overline{\deg(V)}$	$\sigma(\deg(V))$	$\overline{\lvert e\rvert}$	$\sigma(\lvert e\rvert)$
	dated-10-11-u	629 461	141 860	1 429 872	2.272	0.935	10.080	4.955
	dated-10-17-u	1 070 757	229 544	2 471 122	2.308	0.920	10.765	6.790
	gss-19-s100	94 548	31 435	222 806	2.357	0.480	7.088	6.516
	hwmcc10-timeframe-expansion-k45-pdtvisns3p02-tseitin	488 120	163 622	1 138 944	2.333	0.471	6.961	15.164
	itox-vc1130	441 729	152 256	1 143 974	2.590	0.537	7.513	47.981
	manol-pipe-c8nidw	799 867	269 048	1 866 355	2.333	0.471	6.937	16.726
	manol-pipe-g10bid-i	792 175	266 405	1 848 407	2.333	0.471	6.938	21.682
Literal	6s133	96 430	140 968	328 924	3.411	8.176	2.333	0.471
	6s153	171 292	245 440	572 692	3.343	5.838	2.333	0.471
	6s184	66 730	97 516	227 536	3.410	8.468	2.333	0.471
	6s9	68 634	100 384	234 228	3.413	8.427	2.333	0.471
	aaai10-planning-ipc5-pathways-17-step21	107 838	308 235	690 466	6.403	6.230	2.240	1.884
	ACG-20-5p0	649 432	1 390 931	3 269 132	5.034	4.873	2.350	0.923
	ACG-20-5p1	662 392	1 416 850	3 333 531	5.033	4.859	2.353	0.917
	AProVE07-27	15 458	29 194	77 124	4.989	19.416	2.642	1.429
	atco-enc1-opt2-05-4	28 738	386 163	1 652 800	57.513	130.721	4.280	1.364
	atco-enc1-opt2-10-16	18 930	152 744	641 139	33.869	80.832	4.197	1.582
	atco-enc2-opt1-05-21	112 732	526 872	2 097 393	18.605	72.377	3.981	1.538
	atco-enc2-opt1-15-100	117 116	580 963	2 227 755	19.022	70.295	3.835	1.534
	bob12s02	52 588	77 920	181 812	3.457	4.082	2.333	0.471
	countbitssrl032	37 213	55 724	130 020	3.494	5.242	2.333	0.471
	dated-10-11-u	283 720	629 461	1 429 872	5.040	3.081	2.272	0.935
	dated-10-17-u	459 088	1 070 757	2 471 122	5.383	3.851	2.308	0.920
	gss-19-s100	62 870	94 548	222 806	3.544	3.294	2.357	0.480
	hwmcc10-timeframe-expansion-k45-pdtvisns3p02-tseitin	327 243	488 120	1 138 944	3.480	7.599	2.333	0.471
	itox-vc1130	294 326	441 729	1 143 974	3.887	24.408	2.590	0.537
	manol-pipe-c8nidw	538 096	799 867	1 866 355	3.468	8.378	2.333	0.471
	manol-pipe-g10bid-i	532 810	792 175	1 848 407	3.469	10.853	2.333	0.471
SPM	2cubes-sphere	101 492	101 492	1 647 264	16.231	2.654	16.231	2.654
	2D-54019-highK	54 019	54 019	996 414	18.446	3.109	18.446	6.922
	af-shell1	504 855	504 855	17 588 875	34.840	1.275	34.840	1.275
	Andrews	60 000	60 000	760 154	12.669	3.414	12.669	3.414
	as-caida	31 379	26 475	106 762	3.402	30.691	4.033	33.374
	av41092	41 092	41 092	1 683 902	40.979	96.937	40.979	167.038
	BenElechi1	245 874	245 874	13 150 496	53.485	2.995	53.485	2.995
	case39	40 216	40 216	1 042 160	25.914	316.226	25.914	316.226
	ckt11752-dc-1	49 702	49 702	333 029	6.701	23.529	6.701	23.221
	cnr-2000	325 557	247 501	3 216 152	9.879	218.496	12.995	22.679
	denormal	89 400	89 400	1 156 224	12.933	0.474	12.933	0.474
	gearbox	153 746	153 746	9 080 404	59.061	15.410	59.061	15.410
	hvdc1	24 842	24 842	159 981	6.440	2.936	6.440	3.617
	laminar-duct3D	67 173	67 173	3 833 077	57.063	29.628	57.063	37.896
	lhr14	14 270	14 270	307 858	21.574	15.983	21.574	26.269
	light-in-tissue	29 282	29 282	406 084	13.868	2.733	13.868	2.733
	Lin	256 000	256 000	1 766 400	6.900	0.310	6.900	0.310
	lp-pds20	108 175	33 798	232 647	2.151	0.416	6.883	6.162
	m14b	214 765	214 765	3 358 036	15.636	3.131	15.636	3.131
	mc2depi	525 825	525 825	2 100 225	3.994	0.076	3.994	0.076
	mult-dcop-01	25 187	25 187	193 276	7.674	144.207	7.674	143.814
	opt1	15 449	15 449	1 930 655	124.970	42.495	124.970	42.495
	poisson3Db	85 623	85 623	2 374 949	27.737	14.712	27.737	14.712

Table 10: Overview of hypergraph instances in training set.

A.2. Hypergraph Benchmark Set

Overview of all hypergraph instances used for benchmarking. These instances are not part of any training or tuning but only used for evaluation purposes. $\overline{\deg(V)}$ denotes the average hypernode degrees and $\sigma(\deg(V))$ the standard deviations of node degrees. Similarly, $\overline{|e|}$ denotes the average net sizes and $\sigma(|e|)$ the standard deviation of net sizes.

Type	Hypergraph	n	m	p	$\overline{\deg(V)}$	$\sigma(\deg(V))$	$\overline{\lvert e\rvert}$	$\sigma(\lvert e\rvert)$
DAC12	superblue11	952 507	935 731	3 069 269	3.222	6.915	3.280	10.519
	superblue12	1 291 931	1 293 436	4 773 600	3.695	2.145	3.691	20.938
	superblue16	698 339	697 458	2 280 417	3.265	6.059	3.270	9.052
	superblue2	1 010 321	990 899	3 227 167	3.194	5.547	3.257	10.777
	superblue7	1 360 217	1 340 418	4 931 418	3.625	3.099	3.679	16.762
ISPD98	ibm01	12 752	14 111	50 566	3.965	2.329	3.583	3.343
	ibm02	19 601	19 584	81 199	4.143	2.292	4.146	5.452
	ibm03	23 136	27 401	93 573	4.044	3.448	3.415	3.107
	ibm07	45 926	48 117	175 639	3.824	2.415	3.650	3.049
	ibm09	53 395	60 902	222 088	4.159	3.223	3.647	3.133
	ibm11	70 558	81 454	280 786	3.980	3.173	3.447	2.599
	ibm13	84 199	99 666	357 075	4.241	3.342	3.583	3.008
	ibm15	161 570	186 608	715 823	4.430	3.286	3.836	3.510
	ibm17	185 495	189 581	860 036	4.636	2.494	4.537	4.071
Primal	6s10	33 900	99 184	231 428	6.827	16.709	2.333	0.471
	6s11-opt	33 276	97 312	227 060	6.824	15.902	2.333	0.471
	6s12	34 033	99 580	232 352	6.827	16.688	2.333	0.471
	6s130-opt	49 327	144 361	336 841	6.829	13.086	2.333	0.471
	6s131-opt	49 282	144 226	336 526	6.829	13.078	2.333	0.471
	6s16	31 483	91 888	214 404	6.810	17.301	2.333	0.471
	9dlx-vliw-at-b-iq3	69 789	968 295	2 788 367	39.954	224.993	2.880	5.434
	AProVE07-01	7502	28 770	76 290	10.169	13.742	2.652	5.131
	atco-enc1-opt1-05-21	59 517	561 784	2 167 217	36.413	135.808	3.858	1.564
	atco-enc1-opt1-10-21	46 993	270 831	922 875	19.639	70.064	3.408	1.607
	atco-enc1-opt1-15-240	61 642	644 099	2 385 303	38.696	132.336	3.703	1.518
	atco-enc1-opt2-10-12	9495	147 853	618 608	65.151	143.223	4.184	1.587
	atco-enc2-opt1-15-100	58 752	580 963	2 227 755	37.918	134.900	3.835	1.534
	bob12m09-opt	51 144	152 446	355 706	6.955	19.566	2.333	0.471
	c10bi-i	133 998	398 467	929 755	6.939	24.655	2.333	0.471
	ctl-3791-556-unsat-pre	8806	90 812	331 537	37.649	24.330	3.651	0.705
	ctl-4291-567-5-unsat-pre	15 232	134 756	462 322	30.352	23.184	3.431	0.788
	gss-18-s100	31 364	94 269	222 003	7.078	6.495	2.355	0.480
	gss-20-s100	31 503	94 748	223 300	7.088	6.487	2.357	0.480
	gss-22-s100	31 616	95 110	224 220	7.092	6.488	2.357	0.481
	manol-pipe-c10nid-i	252 516	750 877	1 752 045	6.938	21.824	2.333	0.471
Dual	6s10	99 184	33 900	231 428	2.333	0.471	6.827	16.709
	6s11-opt	97 312	33 276	227 060	2.333	0.471	6.824	15.902
	6s12	99 580	34 033	232 352	2.333	0.471	6.827	16.688
	6s130-opt	144 361	49 327	336 841	2.333	0.471	6.829	13.086
	6s131-opt	144 226	49 282	336 526	2.333	0.471	6.829	13.078
	6s16	91 888	31 483	214 404	2.333	0.471	6.810	17.301
	9dlx-vliw-at-b-iq3	968 295	69 789	2 788 367	2.880	5.434	39.954	224.993
	AProVE07-01	28 770	7502	76 290	2.652	5.131	10.169	13.742
	atco-enc1-opt1-05-21	561 784	59 517	2 167 217	3.858	1.564	36.413	135.808
	atco-enc1-opt1-10-21	270 831	46 993	922 875	3.408	1.607	19.639	70.064
	atco-enc1-opt1-15-240	644 099	61 642	2 385 303	3.703	1.518	38.696	132.336
	atco-enc1-opt2-10-12	147 853	9495	618 608	4.184	1.587	65.151	143.223
	atco-enc2-opt1-15-100	580 963	58 752	2 227 755	3.835	1.534	37.918	134.900
	bob12m09-opt	152 446	51 144	355 706	2.333	0.471	6.955	19.566
	c10bi-i	398 467	133 998	929 755	2.333	0.471	6.939	24.655
	ctl-3791-556-unsat-pre	90 812	8806	331 537	3.651	0.705	37.649	24.330
	ctl-4291-567-5-unsat-pre	134 756	15 232	462 322	3.431	0.788	30.352	23.184
	gss-18-s100	94 269	31 364	222 003	2.355	0.480	7.078	6.495
	gss-20-s100	94 748	31 503	223 300	2.357	0.480	7.088	6.487
	gss-22-s100	95 110	31 616	224 220	2.357	0.481	7.092	6.488
	manol-pipe-c10nid-i	750 877	252 516	1 752 045	2.333	0.471	6.938	21.824
Literal	6s10	67 800	99 184	231 428	3.413	8.369	2.333	0.471
	6s11-opt	66 552	97 312	227 060	3.412	7.966	2.333	0.471
	6s12	68 066	99 580	232 352	3.414	8.359	2.333	0.471
	6s130-opt	98 654	144 361	336 841	3.414	6.562	2.333	0.471
	6s131-opt	98 564	144 226	336 526	3.414	6.558	2.333	0.471
	6s16	62 966	91 888	214 404	3.405	8.664	2.333	0.471
	9dlx-vliw-at-b-iq3	139 578	968 295	2 788 367	19.977	112.962	2.880	5.434
	AProVE07-01	15 004	28 770	76 290	5.085	8.516	2.652	5.131
	atco-enc1-opt1-05-21	118 700	561 784	2 167 217	18.258	70.532	3.858	1.564
	atco-enc1-opt1-10-21	93 632	270 831	922 875	9.856	38.866	3.408	1.607
	atco-enc1-opt1-15-240	122 885	644 099	2 385 303	19.411	69.089	3.703	1.518
	atco-enc1-opt2-10-12	18 634	147 853	618 608	33.198	81.246	4.184	1.587

Table 11: Overview of hypergraph instances in benchmark set.

Type	Hypergraph	n	m	p	$\overline{\deg(V)}$	$\sigma(\deg(V))$	$\overline{\lvert e \rvert}$	$\sigma(\lvert e \rvert)$
	atco-enc2-opt1-15-100	117 116	580 963	2 227 755	19.022	70.295	3.835	1.534
	bob12m09-opt	102 288	152 446	355 706	3.477	9.795	2.333	0.471
	c10bi-i	267 996	398 467	929 755	3.469	12.338	2.333	0.471
	ctl-3791-556-unsat-pre	17 612	90 812	331 537	18.825	12.210	3.651	0.705
	ctl-4291-567-5-unsat-pre	30 464	134 756	462 322	15.176	11.637	3.431	0.788
	gss-18-s100	62 728	94 269	222 003	3.539	3.284	2.355	0.480
	gss-20-s100	63 006	94 748	223 300	3.544	3.280	2.357	0.480
	gss-22-s100	63 232	95 110	224 220	3.546	3.280	2.357	0.481
	manol-pipe-c10nid-i	505 032	750 877	1 752 045	3.469	10.923	2.333	0.471
SPM	c-61	43 618	43 618	310 016	7.108	16.760	7.108	16.760
	cfd1	70 656	70 656	1 828 364	25.877	2.972	25.877	2.972
	Ill-Stokes	20 896	20 896	191 368	9.158	1.562	9.158	1.644
	Maragal-6	10 152	21 251	537 694	52.964	54.574	25.302	202.891
	mixtank-new	29 957	29 957	1 995 041	66.597	38.335	66.597	38.335
	Oregon-1	11 492	11 174	46 818	4.074	32.641	4.190	33.095
	powersim	15 838	15 838	67 562	4.266	3.421	4.266	2.701
	Pres-Poisson	14 822	14 822	715 804	48.293	5.117	48.293	5.117
	rajat26	51 032	51 032	249 302	4.885	22.404	4.885	22.760
	Reuters911	13 332	13 314	296 076	22.208	66.741	22.238	66.781
	RFdevice	74 104	74 104	365 580	4.933	0.416	4.933	1.782
	rgg-n-2-18-s0	262 144	262 141	3 094 566	11.805	3.449	11.805	3.448
	rim	22 560	22 560	1 014 951	44.989	25.979	44.989	26.576
	scircuit	170 998	170 998	958 936	5.608	4.392	5.608	4.392
	sme3Db	29 067	29 067	2 081 063	71.595	37.067	71.595	37.066
	spmsrtls	29 995	29 995	229 947	7.666	0.473	7.666	0.473
	ted-A	10 605	10 605	424 587	40.037	22.782	40.037	37.196
	thermal1	82 654	82 654	574 458	6.950	0.877	6.950	0.877
	thermomech-TC	102 158	102 158	711 558	6.965	0.715	6.965	0.715
	trans4	116 835	116 835	766 396	6.560	361.435	6.560	361.498
	vibrobox	12 328	12 328	342 828	27.809	16.089	27.809	16.089
	viscoplastic2	32 769	32 769	381 326	11.637	14.439	11.637	13.957
	Zhao2	33 861	33 861	166 453	4.916	1.038	4.916	0.437

Table 11: Overview of hypergraph instances in benchmark set.

A.3. List of Features

List of features regarding to a hypergraph $H = (V, E, c, \omega)$ and a pair of hypernodes (u, v) with $u \neq v$, $u, v \in e$ for any $e \in E$. The first eleven features are global features that are computed once per hypergraph instance, whereas the subsequent 14 features are local features.

F01 Count of hypernodes n

F02 Count of hyperedges m

F03 Count of pins p

F04 Network ratio

$$r(H) := \frac{p - m}{n} \tag{A.1}$$

F05 Standard deviation of hypernode degrees

F06 Minimum hypernode degree

F07 Maximum hypernode degree

F08 First-quartile of hypernode degrees

F09 Average hyperedge size

F10 Standard deviation of hyperedge sizes

F11 Maximum hyperedge size

F12 Count of common neighbours $|\,\Gamma(u) \cap \Gamma(v)\,|$

F13 Count of all neighbours $|\Gamma(u) \cup \Gamma(v)|$

F14 Jaccard indices

$$J(u,v) := \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} \tag{A.2}$$

F15 Dice similarity

$$D(u,v) := \frac{2\,|\Gamma(u) \cap \Gamma(v)|}{\sum_{w \in \Gamma(u) \cap \Gamma(v)} \deg(w)} \tag{A.3}$$

F16 Cosine similarity

$$C(u,v) := \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{\deg(u)\deg(v)}} \tag{A.4}$$

F17 Average hypernode degrees of u and v

$$\frac{\deg(u) + \deg(v)}{2} \tag{A.5}$$

F18 Average hypernode degree of common neighbours

$$\frac{\sum_{w \in \Gamma(u) \cap \Gamma(v)} \deg(w)}{|\Gamma(u) \cap \Gamma(v)|} \tag{A.6}$$

F19 χ^2-*metric* hypernode degree of common neighbours

$$\chi^2_{deg,\cap}(u,v) := \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{\left(\deg(w) - \overline{\deg(V)}\right)^2}{\overline{\deg(V)}} \tag{A.7}$$

F20 Average hypernode degree of all neighbours

$$\frac{\sum_{w \in \Gamma(u) \cup \Gamma(v)} \deg(w)}{|\Gamma(u) \cup \Gamma(v)|} \tag{A.8}$$

F21 χ^2-*metric* hypernode degree of all neighbours

$$\chi^2_{deg,\cup}(u,v) := \sum_{w \in \Gamma(u) \cup \Gamma(v)} \frac{\left(\deg(w) - \overline{\deg(V)}\right)^2}{\overline{\deg(V)}} \tag{A.9}$$

F22 Closeness metric within the HGCEP algorithm [68]

$$\text{closeness}(u,v) := \frac{|I(u) \cap I(v)|}{\min(\deg(u), \deg(v))} \ . \tag{A.10}$$

F23 Bandwidth clustering rating function [58]

$$\Psi(u,v) := \sum_{e \in I(u) \cap I(v)} \frac{1}{|e| - 1} \tag{A.11}$$

F24 Strawman connectivity function [31, 65]

$$\text{connectivity}(u,v) := \frac{\Psi(u,v)}{(\deg(u) - \Psi(u,v))\,(\deg(v) - \Psi(u,v))} \tag{A.12}$$

F25 Count of common incident nets $|I(u) \cap I(v)|$

A.4. Training Set Feature Correlation

Table 12 shows the correlation matrix of the generated training samples. Due to the symmetry of the matrices, the lower half has been omitted.

A.5. Local Feature Value Distributions of Training Set

Fig. 8 shows the distribution of the generated sample feature values regarding all 14 local features (**F12** – **F25**) on the training set given in Section A.1.

A.6. Principal Components

Table 13 shows the calculated principal components with respective eigenvalues in decreasing order. Only the first 20 components are given because they already explain almost all variance on the data. Thereafter, a plot showing the (cumulative) explained variance regarding the principal components is given in Fig. 9.

A.7. Trained Model

Table 15-18 show the final weights that have been trained by the model presented. Configurations for model training were χ_2, χ_4, χ_8, and χ_{16}. In each table, the first column shows the trained weights in respect to the principal component variables, whereas the remaining columns show the (sorted) weights regarding to the actual features. These weights have been calculated by the formulas given in Section 4.4.3 from the principal component weights. Besides the weights given below, the trained biases are depicted in Table 14.

Trained Weights		PC × Weight		Sorted Weights	
PC1	−1.074 031	F01	−2.377 531	F06	−3.780 470
PC2	−1.057 926	F02	−0.827 558	F05	−2.466 181
PC3	0.172 668	F03	−0.118 051	F01	−2.377 531
PC4	−0.793 658	F04	−1.971 080	F25	−2.256 006
PC5	0.862 914	F05	−2.466 181	F04	−1.971 080
PC6	1.244 527	F06	−3.780 470	F12	−1.861 438
PC7	1.984 199	F07	−0.546 643	F16	−1.449 878
PC8	0.132 208	F08	−0.556 076	F02	−0.827 558
PC9	−3.337 685	F09	−0.305 025	F15	−0.715 931
PC10	1.214 255	F10	−0.069 950	F08	−0.556 076
PC11	−4.005 485	F11	0.048 892	F07	−0.546 643
PC12	0.455 890	F12	−1.861 438	F09	−0.305 025
PC13	0.102 353	F13	0.035 988	F23	−0.151 894
PC14	1.733 614	F14	0.430 530	F03	−0.118 051
PC15	−1.161 052	F15	−0.715 931	F10	−0.069 950
PC16	0.382 953	F16	−1.449 878	F13	0.035 988
PC17	1.239 500	F17	0.707 782	F11	0.048 892
PC18	−0.143 975	F18	0.485 509	F20	0.099 216

Table 15: Trained model weights θ for configuration χ_2.

Trained Weights		PC × Weight		Sorted Weights	
PC19	0.624 102	F19	0.357 077	F19	0.357 077
PC20	2.503 893	F20	0.099 216	F14	0.430 530
		F21	0.895 118	F18	0.485 509
		F22	1.927 743	F17	0.707 782
		F23	−0.151 894	F21	0.895 118
		F24	1.638 310	F24	1.638 310
		F25	−2.256 006	F22	1.927 743

Table 15: Trained model weights θ for configuration χ_2.

Trained Weights		PC × Weight		Sorted Weights	
PC1	−1.140 626	F01	−2.420 451	F06	−4.451 649
PC2	−0.880 468	F02	−0.612 157	F05	−2.497 803
PC3	0.156 043	F03	−0.069 827	F25	−2.433 747
PC4	−0.703 956	F04	−1.734 019	F01	−2.420 451
PC5	1.055 820	F05	−2.497 803	F12	−1.759 176
PC6	1.580 498	F06	−4.451 649	F04	−1.734 019
PC7	1.401 284	F07	−0.750 298	F16	−0.872 869
PC8	0.728 773	F08	−0.340 745	F07	−0.750 298
PC9	−3.471 314	F09	−0.168 370	F02	−0.612 157
PC10	1.243 087	F10	−0.282 811	F11	−0.428 755
PC11	−4.264 803	F11	−0.428 755	F08	−0.340 745
PC12	0.174 213	F12	−1.759 176	F15	−0.309 094
PC13	0.476 610	F13	0.153 572	F10	−0.282 811
PC14	1.267 631	F14	0.664 590	F09	−0.168 370
PC15	−0.653 063	F15	−0.309 094	F03	−0.069 827
PC16	0.560 719	F16	−0.872 869	F23	0.005 142
PC17	1.693 241	F17	0.925 816	F19	0.150 614
PC18	−0.095 501	F18	0.388 761	F13	0.153 572
PC19	0.773 145	F19	0.150 614	F18	0.388 761
PC20	2.788 756	F20	0.436 655	F20	0.436 655
		F21	0.911 352	F14	0.664 590
		F22	2.069 115	F21	0.911 352
		F23	0.005 142	F17	0.925 816
		F24	1.182 116	F24	1.182 116
		F25	−2.433 747	F22	2.069 115

Table 16: Trained model weights θ for configuration χ_4.

Trained Weights		PC × Weight		Sorted Weights	
PC1	−1.106 286	F01	−2.095 138	F06	−4.215 220
PC2	−0.730 284	F02	−0.517 339	F25	−2.513 985
PC3	0.223 768	F03	0.445 331	F05	−2.227 685
PC4	−0.766 722	F04	−1.541 891	F01	−2.095 138

Table 17: Trained model weights θ for configuration χ_8.

Trained Weights		PC × Weight		Sorted Weights	
PC5	0.645 064	F05	−2.227 685	F04	−1.541 891
PC6	1.517 715	F06	−4.215 220	F12	−1.430 784
PC7	0.072 785	F07	−1.092 341	F07	−1.092 341
PC8	1.494 539	F08	−0.400 105	F10	−1.024 801
PC9	−3.398 498	F09	−0.765 670	F09	−0.765 670
PC10	1.228 565	F10	−1.024 801	F11	−0.532 017
PC11	−3.757 705	F11	−0.532 017	F02	−0.517 339
PC12	−0.173 322	F12	−1.430 784	F08	−0.400 105
PC13	0.843 739	F13	0.348 004	F15	−0.196 468
PC14	1.537 078	F14	0.864 251	F19	−0.080 809
PC15	0.144 702	F15	−0.196 468	F16	−0.044 116
PC16	0.629 818	F16	−0.044 116	F23	0.142 590
PC17	1.471 114	F17	1.077 300	F24	0.276 164
PC18	−0.019 224	F18	0.292 452	F18	0.292 452
PC19	0.607 581	F19	−0.080 809	F13	0.348 004
PC20	3.003 989	F20	1.138 310	F03	0.445 331
		F21	0.877 965	F14	0.864 251
		F22	1.919 592	F21	0.877 965
		F23	0.142 590	F17	1.077 300
		F24	0.276 164	F20	1.138 310
		F25	−2.513 985	F22	1.919 592

Table 17: Trained model weights θ for configuration χ_8.

Trained Weights		PC × Weight		Sorted Weights	
PC1	−1.051 619	F01	−1.873 995	F06	−3.941 943
PC2	−0.561 281	F02	−0.602 066	F25	−2.573 413
PC3	0.186 868	F03	0.811 416	F05	−1.936 295
PC4	−0.835 684	F04	−1.349 845	F01	−1.873 995
PC5	0.448 364	F05	−1.936 295	F07	−1.403 015
PC6	1.468 040	F06	−3.941 943	F10	−1.389 111
PC7	−0.729 113	F07	−1.403 015	F04	−1.349 845
PC8	1.721 431	F08	−0.468 159	F09	−1.149 443
PC9	−3.339 197	F09	−1.149 443	F12	−1.145 125
PC10	1.264 178	F10	−1.389 111	F11	−0.669 732
PC11	−3.404 449	F11	−0.669 732	F02	−0.602 066
PC12	−0.347 248	F12	−1.145 125	F08	−0.468 159
PC13	0.953 824	F13	0.576 435	F24	−0.300 447
PC14	1.736 106	F14	0.938 662	F19	−0.236 993
PC15	0.843 317	F15	−0.143 804	F15	−0.143 804
PC16	0.593 659	F16	0.349 864	F23	0.115 182
PC17	1.245 351	F17	1.181 380	F16	0.349 864
PC18	0.053 798	F18	0.352 108	F18	0.352 108
PC19	0.420 875	F19	−0.236 993	F13	0.576 435
PC20	3.050 799	F20	1.434 695	F03	0.811 416

Table 18: Trained model weights θ for configuration χ_{16}.

Trained Weights	PC × Weight		Sorted Weights	
	F21	0.879 427	F21	0.879 427
	F22	1.777 372	F14	0.938 662
	F23	0.115 182	F17	1.181 380
	F24	−0.300 447	F20	1.434 695
	F25	−2.573 413	F22	1.777 372

Table 18: Trained model weights θ for configuration χ_{16}.

A.8. Hypergraph Pruning Solution Quality Plots

Fig. 10 comprises performance profile plots for all configurations aggregated as well as for each configuration on its own.

A.9. Hypergraph Pruning Runtime Plots

Fig. 11 and 12 contain runtime plots both for all configurations aggregated as well as for each configuration on its own. The plots on the left show absolute running times of the presented approach as well as of the partitioner KaHyPar-CA, whereas the plots on the right show the running times of our approach in relation to the running times of KaHyPar-CA per-instance.

	F01	F02	F03	F04	F05	F06	F07	F08	F09	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	F25
F01	1.00	0.31	0.04	−0.45	−0.32	−0.07	−0.09	−0.29	−0.27	−0.07	0.14	−0.10	−0.10	−0.20	0.60	0.00	−0.07	−0.13	−0.04	−0.22	−0.06	0.05	−0.02	0.18	−0.03
F02		1.00	0.02	−0.33	−0.10	−0.04	−0.05	−0.27	−0.47	−0.32	0.02	−0.15	−0.16	−0.35	−0.01	−0.11	−0.06	−0.04	−0.06	−0.06	−0.07	−0.30	−0.03	0.16	−0.05
F03			1.00	0.49	−0.23	0.57	−0.17	0.70	0.45	−0.22	−0.22	−0.10	−0.12	0.62	−0.40	−0.10	−0.06	−0.05	−0.05	−0.00	−0.07	0.67	−0.02	−0.17	−0.02
F04				1.00	0.18	0.51	−0.13	0.86	0.76	−0.04	−0.21	−0.01	−0.04	0.62	−0.57	−0.10	−0.01	0.09	−0.04	0.23	−0.04	0.50	−0.02	−0.23	−0.00
F05					1.00	−0.12	0.71	−0.19	−0.11	0.28	0.30	0.50	0.54	0.04	−0.25	0.09	0.44	0.35	0.40	0.37	0.47	−0.27	0.23	−0.08	0.26
F06						1.00	−0.14	0.57	0.46	−0.11	−0.17	−0.06	−0.08	0.41	−0.32	−0.07	−0.06	−0.03	−0.05	0.03	−0.06	0.43	−0.03	−0.14	−0.02
F07							1.00	−0.20	−0.14	0.28	0.47	0.60	0.65	−0.01	−0.07	0.21	0.51	0.26	0.51	0.16	0.60	−0.13	0.30	0.01	0.33
F08								1.00	0.80	−0.14	−0.24	−0.08	−0.11	0.66	−0.49	−0.10	−0.08	−0.03	−0.08	0.07	−0.09	0.69	−0.04	−0.20	−0.03
F09									1.00	0.36	−0.16	0.05	0.04	0.63	−0.34	0.07	−0.04	−0.02	−0.04	0.06	−0.04	0.62	−0.02	−0.22	−0.00
F10										1.00	0.45	0.53	0.58	0.11	0.08	0.38	0.13	0.06	0.12	0.02	0.24	−0.02	0.06	−0.05	0.10
F11											1.00	0.55	0.59	−0.06	0.19	0.34	0.19	0.06	0.18	−0.03	0.31	−0.09	0.09	0.16	0.12
F12												1.00	0.89	0.18	−0.06	0.42	0.66	−0.01	0.52	−0.00	0.71	0.00	0.47	−0.05	0.58
F13													1.00	0.09	−0.06	0.36	0.67	0.15	0.56	−0.01	0.77	−0.05	0.38	−0.05	0.46
F14														1.00	−0.36	0.12	0.08	−0.04	0.06	0.13	0.07	0.72	0.08	−0.29	0.10
F15															1.00	0.08	−0.08	−0.15	−0.07	−0.24	−0.07	−0.13	−0.04	0.14	−0.05
F16																1.00	−0.02	−0.03	0.05	−0.04	0.10	−0.01	−0.01	−0.04	−0.01
F17																	1.00	0.11	0.78	0.03	0.89	−0.03	0.69	−0.04	0.78
F18																		1.00	0.27	0.53	0.18	−0.11	−0.01	−0.05	−0.01
F19																			1.00	0.13	0.90	−0.03	0.46	−0.03	0.59
F20																				1.00	−0.01	−0.06	−0.01	−0.07	−0.00
F21																					1.00	−0.03	0.57	−0.03	0.65
F22																						1.00	0.06	−0.02	0.08
F23																							1.00	−0.02	0.88
F24																								1.00	−0.02
F25																									1.00

Table 12: Correlation matrix of the generated training sample features

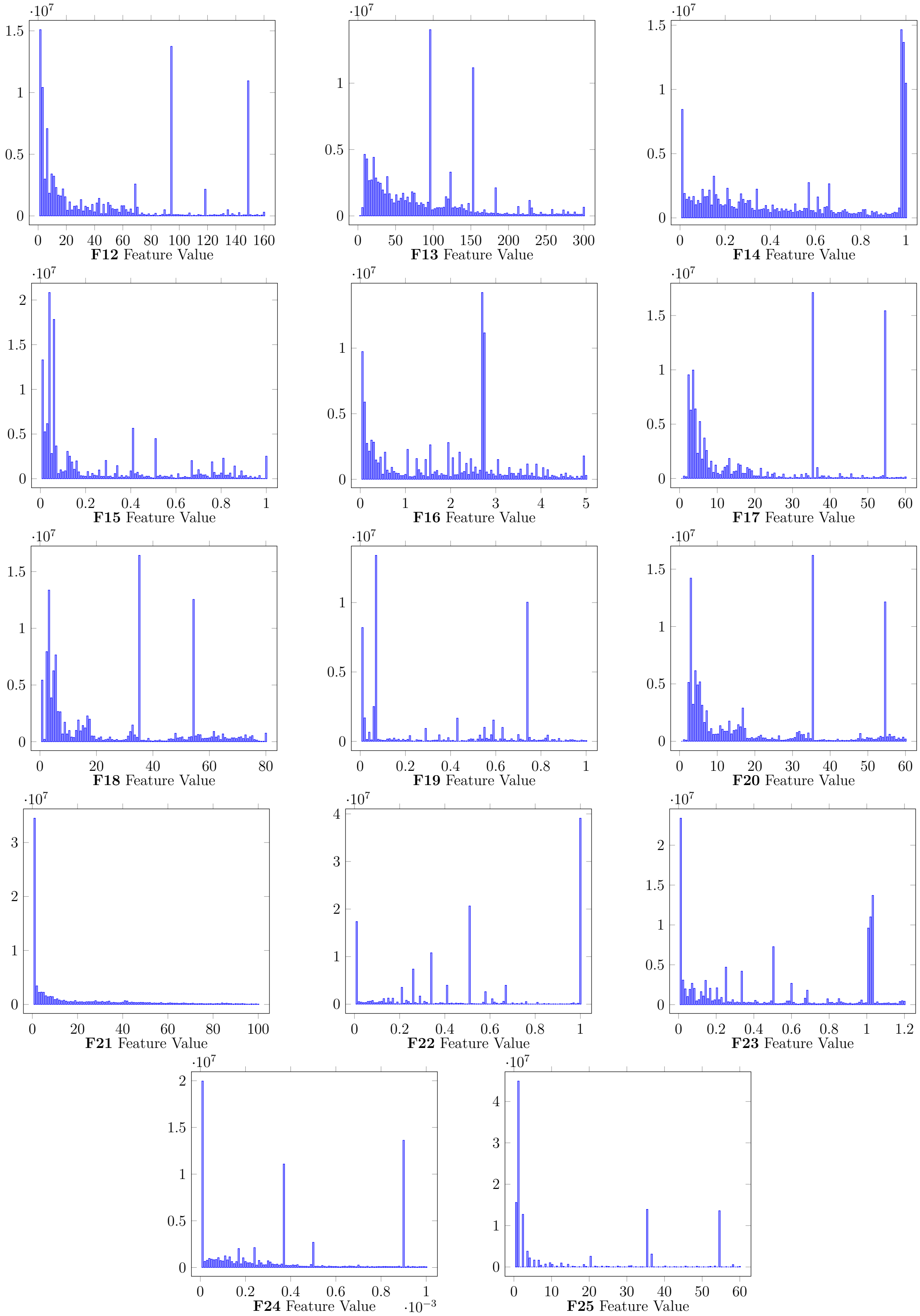


Figure 8: Feature value distributions for all 14 local features.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Eigen value	6.30	5.26	2.30	2.24	1.34	1.14	0.96	0.78	0.69	0.67	0.54	0.47	0.40	0.42	0.30	0.28	0.24	0.17	0.15	0.11
Explained Variance (in %)	25.22	21.06	9.21	8.95	5.34	4.55	3.85	3.11	2.78	2.69	2.17	1.87	1.61	1.70	1.19	1.11	0.97	0.68	0.59	0.42
Cumulative Explained Variance (in %)	25.22	46.27	55.49	64.44	69.78	74.33	78.18	81.29	84.06	86.75	88.92	90.79	92.39	94.09	95.28	96.40	97.36	98.04	98.63	99.06
PCA components	0.03	−0.18	−0.31	−0.21	−0.39	0.31	0.17	0.01	0.04	0.19	−0.00	−0.24	0.29	−0.01	−0.19	−0.29	−0.35	0.35	−0.06	−0.06
	0.04	−0.18	−0.17	0.19	−0.43	−0.40	0.12	−0.20	−0.20	0.18	0.29	−0.16	0.27	0.10	0.15	0.35	0.30	−0.03	0.02	0.00
	0.12	0.30	−0.20	−0.00	−0.35	−0.08	0.13	0.01	0.07	−0.07	0.18	−0.01	−0.18	−0.23	−0.14	−0.32	−0.09	−0.63	0.11	0.11
	0.07	0.38	0.12	0.09	0.05	−0.05	−0.13	0.03	0.05	0.11	−0.07	0.03	0.42	0.31	−0.23	0.02	−0.04	−0.05	−0.11	−0.04
	−0.26	0.04	0.31	0.19	−0.06	−0.09	−0.03	0.06	0.37	0.16	−0.17	−0.23	0.29	−0.12	−0.31	0.18	−0.12	−0.24	−0.20	0.08
	0.09	0.26	−0.10	0.01	−0.24	−0.13	0.06	0.07	−0.28	0.34	−0.71	0.02	−0.29	−0.08	0.01	0.16	0.00	0.08	−0.02	0.00
	−0.30	0.01	0.14	0.03	−0.22	−0.07	−0.02	0.17	0.36	−0.04	−0.17	−0.05	0.16	−0.35	0.56	−0.15	0.11	0.13	0.19	0.01
	0.12	0.38	−0.05	−0.01	−0.04	−0.04	−0.09	0.07	−0.03	−0.02	0.04	0.16	0.26	0.23	0.16	−0.23	0.12	−0.01	0.15	−0.16
	0.06	0.36	0.07	−0.19	0.14	0.14	−0.09	0.04	−0.26	0.15	0.09	−0.15	0.29	−0.01	0.27	−0.01	−0.04	0.03	0.18	0.15
	−0.18	0.04	0.26	−0.39	0.12	0.11	−0.03	−0.03	−0.30	0.37	0.26	−0.27	−0.09	−0.26	0.06	0.12	−0.07	−0.18	−0.04	0.01
	−0.21	−0.07	0.13	−0.30	−0.31	−0.10	−0.14	−0.01	0.17	0.21	0.11	0.51	−0.14	0.38	0.15	0.12	−0.35	−0.05	0.02	0.16
	−0.34	0.08	0.01	−0.20	−0.01	−0.17	0.02	−0.11	0.01	0.03	0.01	−0.05	−0.12	0.12	−0.20	−0.19	0.30	0.13	−0.10	−0.31
	−0.35	0.07	0.07	−0.17	−0.07	−0.11	0.02	0.12	−0.10	0.06	0.08	0.06	−0.08	0.02	−0.24	−0.25	0.31	0.08	−0.00	−0.11
	0.01	0.36	−0.02	−0.13	−0.08	0.07	0.12	−0.12	0.31	−0.16	0.15	−0.16	−0.20	−0.02	−0.20	0.50	−0.09	0.25	0.43	−0.14
	−0.00	−0.26	−0.16	−0.28	0.03	0.45	0.09	0.07	0.15	0.09	−0.26	0.03	0.15	0.21	−0.01	0.20	0.42	−0.42	0.21	−0.02
	−0.11	0.00	0.18	−0.37	−0.09	−0.15	0.21	−0.43	−0.22	−0.56	−0.29	−0.01	0.24	−0.00	−0.01	0.00	−0.08	−0.06	−0.02	0.16
	−0.33	0.08	−0.23	0.15	0.08	0.03	0.00	0.10	−0.06	−0.02	0.01	−0.08	−0.03	0.11	−0.16	−0.05	0.11	0.15	0.31	0.62
	−0.09	0.03	0.29	0.28	−0.29	0.43	0.05	−0.01	−0.31	−0.06	0.10	0.48	0.13	−0.31	−0.21	0.11	0.10	0.09	0.06	−0.01
	−0.30	0.07	−0.14	0.17	−0.06	0.18	0.06	0.27	−0.23	−0.28	0.01	−0.17	−0.05	0.21	0.23	0.21	−0.29	−0.17	−0.22	−0.20
	−0.04	0.08	0.32	0.30	−0.20	0.35	0.01	−0.44	0.05	0.07	−0.02	−0.30	−0.29	0.37	0.17	−0.23	0.09	−0.02	−0.00	0.01
	−0.35	0.07	−0.15	0.07	−0.03	0.03	0.03	0.28	−0.16	−0.15	0.03	−0.04	−0.01	0.12	−0.01	0.05	−0.04	−0.06	0.02	−0.17
	0.08	0.31	−0.20	−0.20	−0.16	0.22	−0.13	−0.02	0.22	−0.14	0.11	0.05	−0.06	−0.13	0.12	0.17	0.31	0.12	−0.59	0.12
	−0.24	0.07	−0.32	0.12	0.20	0.05	−0.06	−0.46	0.06	0.16	−0.05	0.20	0.11	−0.21	0.08	0.01	−0.14	−0.11	0.15	−0.42
	−0.00	−0.14	−0.09	−0.04	−0.22	0.01	−0.88	−0.07	−0.12	−0.17	−0.09	−0.17	−0.01	−0.07	−0.14	0.02	−0.00	−0.05	0.14	−0.04
	−0.27	0.09	−0.33	0.11	0.19	0.05	−0.05	−0.35	0.01	0.13	−0.01	0.11	0.03	−0.05	0.03	0.02	−0.04	−0.01	−0.22	0.33

Table 13: Components yielded by the PCA on the training samples.

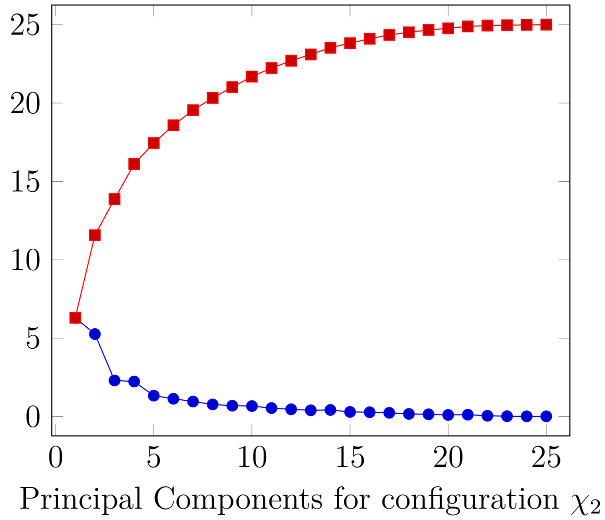
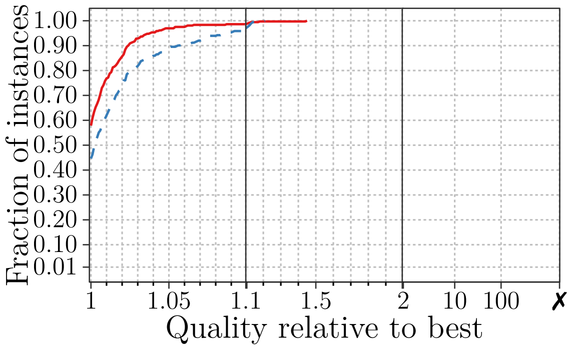


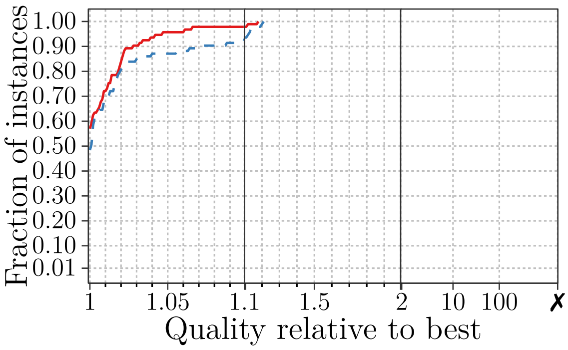
Figure 9: (Cumulative) Explained Variance of the 25 Principal Components on the training samples.

Configuration	Bias θ_0
χ_2	$-1.979\,262$
χ_4	$-2.104\,753$
χ_8	$-1.766\,289$
χ_{16}	$-1.625\,009$

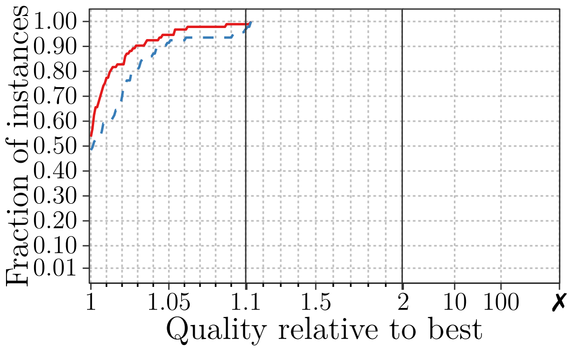
Table 14: Trained model biases θ_0.



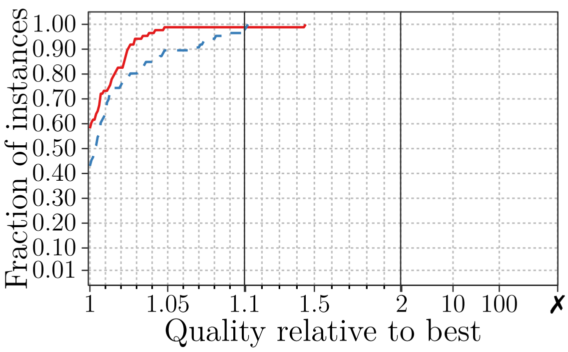
(a) Aggregated plot for all configurations χ.



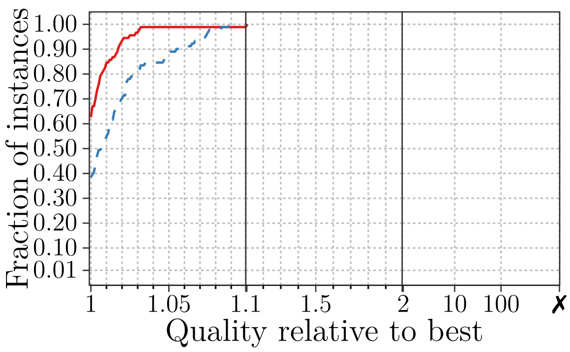
(b) Configuration χ_2.



(c) Configuration χ_4.

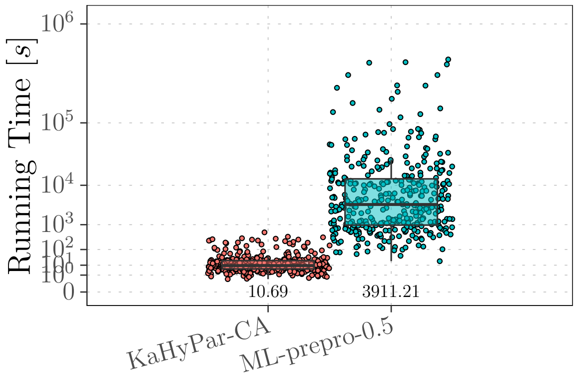


(d) Configuration χ_8.

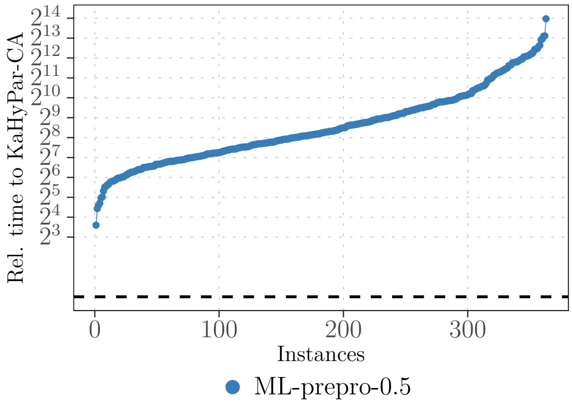


(e) Configuration χ_{16}.

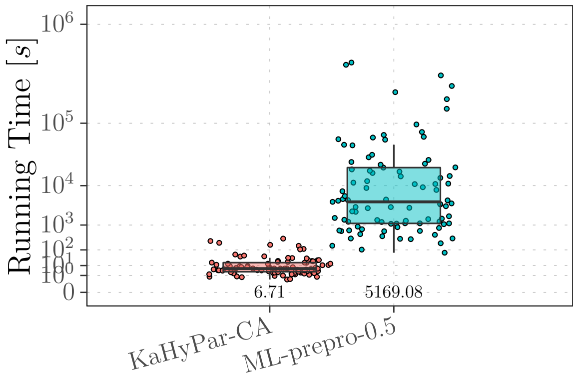
Figure 10: Performance profile plots comparing the presented approach (dashed line) and the KaHyPar-CA partitioner (solid line) in respect of all configurations.



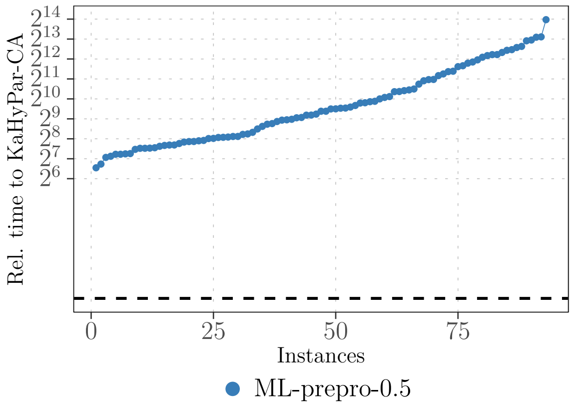
(a) Absolute runtime for all configurations.



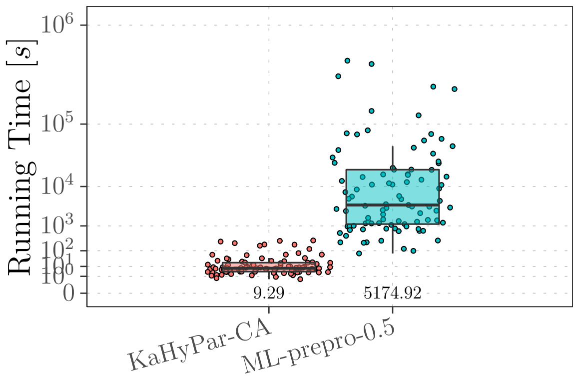
(b) Relative runtime for all configurations.



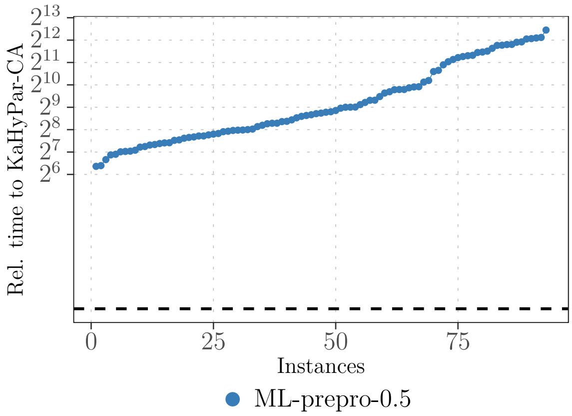
(c) Absolute runtime for configuration χ_2.



(d) Relative runtime for configuration χ_2.

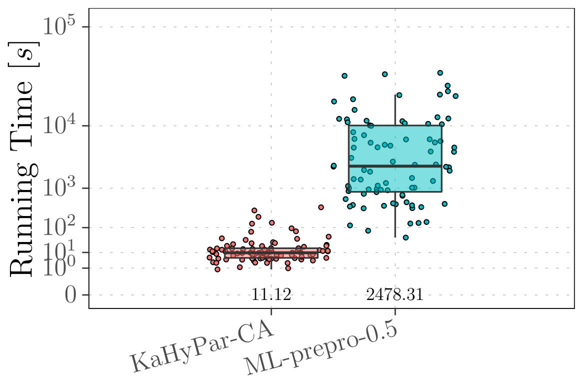


(e) Absolute runtime for configuration χ_4.

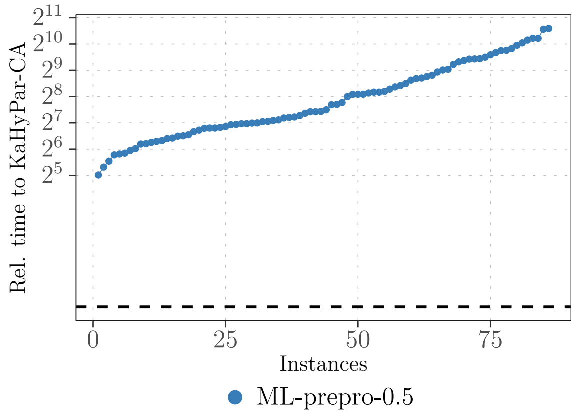


(f) Relative runtime for configuration χ_4.

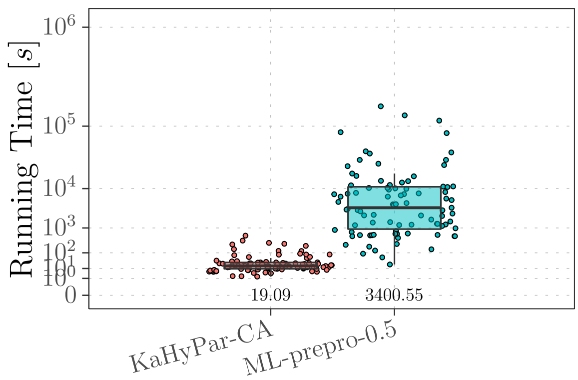
Figure 11: Runtime plots comparing the presented approach and the KaHyPar-CA partitioner in respect of all configurations.



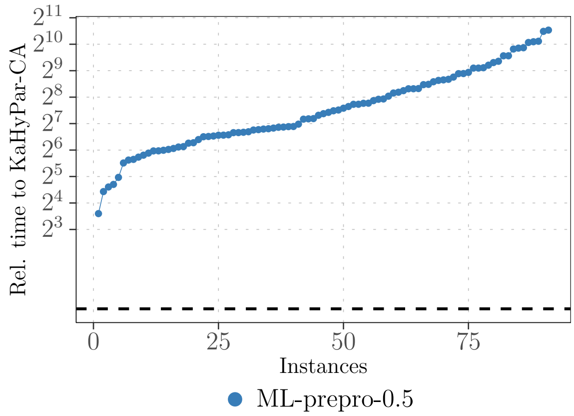
(a) Absolute runtime for configuration χ_8.



(b) Relative runtime for configuration χ_8.



(c) Absolute runtime for configuration χ_{16}.



(d) Relative runtime for configuration χ_{16}.

Figure 12: Runtime plots comparing the presented approach and the KaHyPar-CA partitioner in respect of all configurations (continued).