

Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure

Tobias Heuer¹ and Sebastian Schlag²

1 Karlsruhe Institute of Technology, Karlsruhe, Germany
tobias.heuer@gmx.net

2 Karlsruhe Institute of Technology, Karlsruhe, Germany
sebastian.schlag@kit.edu

Abstract

We present an improved coarsening process for multilevel hypergraph partitioning that incorporates global information about the community structure. Community detection is performed via modularity maximization on a bipartite graph representation. The approach is made suitable for different classes of hypergraphs by defining weights for the graph edges that express structural properties of the hypergraph. We integrate our approach into a leading multilevel hypergraph partitioner with strong local search algorithms and perform extensive experiments on a large benchmark set of hypergraphs stemming from application areas such as VLSI design, SAT solving, and scientific computing. Our results indicate that respecting community structure during coarsening not only significantly improves the solutions found by the initial partitioning algorithm, but also consistently improves overall solution quality.

1998 ACM Subject Classification G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases multilevel hypergraph partitioning, coarsening algorithms, community detection

Digital Object Identifier 10.4230/LIPIcs.SEA.2017.21

1 Introduction

Hypergraphs are a generalization of graphs, where each (hyper)edge (also called *net*) can connect more than two vertices. The k -way hypergraph partitioning problem is the generalization of the well-known graph partitioning problem: partition the vertex set into k disjoint blocks of bounded size (at most $1 + \varepsilon$ times the average block size), while minimizing an objective function defined on the nets. Hypergraph partitioning (HGP) has a wide range of applications. Two prominent areas are VLSI design and scientific computing (e.g. accelerating sparse matrix-vector multiplications) [53]. While the former is an example of a field where small optimizations can lead to significant savings [63], the latter exemplifies problems where hypergraph-based modeling is more flexible than graph-based approaches [16, 34, 35, 36, 43]. HGP also finds application as a preprocessing step in SAT solving, where it is used to identify groups of connected variables [3, 24, 48].

Since hypergraph partitioning is NP-hard [46] and since it is even NP-hard to find good approximate solutions for graphs [14], heuristic *multilevel* algorithms [15, 19, 33, 37] are used in practice. These algorithms consist of three phases: In the *coarsening phase*, the hypergraph is coarsened to obtain a hierarchy of smaller hypergraphs. After applying an *initial partitioning* algorithm to the smallest hypergraph in the second phase, coarsening is undone and, at each level, a *local search* method is used to improve the partition induced by the coarser level.

Coarsening is deemed to be the most important phase of the multilevel paradigm and an area where future research is required to devise algorithms that are suitable for a wide range of hypergraphs [40]. In order to create hypergraphs that are *smaller than* but structurally *similar to* the given hypergraph, coarsening schemes try to identify and merge naturally existing clusters of vertices [22]. State of the art hypergraph partitioning tools compute matchings or clusterings using *local* similarity measures that only take into account the direct neighborhood of each vertex [8, 5, 16, 22, 41, 42, 59, 60, 61]. *Global* considerations are avoided due to the high running times of the respective algorithms [58].

Outline and Contribution. After introducing basic concepts and summarizing related work in Section 2, we present our community-aware hypergraph coarsening framework in Section 3. In a preprocessing phase, we perform modularity maximization on a bipartite graph representation to detect *global* community structure in the input hypergraph. This information is used to guide the coarsening process and to prevent contractions that obscure naturally existing clustering structure. By incorporating information about the net sizes and vertex degrees into the edge weights of the bipartite graph, we make our approach suitable for different classes of hypergraphs. We implemented our algorithm in the open source HGP framework KaHyPar [1]. Extensive experiments presented in Section 4 indicate that respecting community structure during coarsening significantly improves solution quality while having only a moderate impact on the running time. Section 5 concludes the paper.

2 Preliminaries

Notation and Definitions. An *undirected hypergraph* $H = (V, E, c, \omega)$ is defined as a set of n vertices V and a set of m hyperedges/nets E with vertex weights $c : V \rightarrow \mathbb{R}_{>0}$ and net weights $\omega : E \rightarrow \mathbb{R}_{>0}$, where each net is a subset of the vertex set V (i.e., $e \subseteq V$). The vertices of a net are called *pins*. We use P to denote the multiset of all pins in H . We extend c and ω to sets, i.e., $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. A vertex v is *incident* to a net e if $v \in e$. $I(v)$ denotes the set of all incident nets of v . The *degree* of a vertex v is $d(v) := |I(v)|$. The set $\Gamma(v) := \{u \mid \exists e \in E : \{v, u\} \subseteq e\}$ denotes the neighbors of v . The *size* $|e|$ of a net e is the number of its pins. Nets of size one are called *single-vertex* nets. A *k-way partition* of a hypergraph H is a partition of its vertex set into k *blocks* $\Pi = \{V_1, \dots, V_k\}$ such 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 $i \neq j$. We call a *k-way partition* Π ε -*balanced* if each block $V_i \in \Pi$ satisfies the *balance constraint*: $c(V_i) \leq L_{\max} := (1 + \varepsilon) \lceil \frac{c(V)}{k} \rceil$ for some parameter ε . Given a *k-way partition* Π , the number of pins of a net e in block V_i is defined as $\Phi(e, V_i) := |\{v \in V_i \mid v \in e\}|$. For each net e , $\Lambda(e) := \{V_i \mid \Phi(e, V_i) > 0\}$ denotes the *connectivity set* of e . The *connectivity* of a net e is the cardinality of its connectivity set: $\lambda(e) := |\Lambda(e)|$. A net is called *cut net* if $\lambda(e) > 1$. The *k-way hypergraph partitioning problem* is to find an ε -balanced *k-way partition* Π of a hypergraph H that minimizes an objective function over the cut nets for some ε . Several objective functions exist in the literature [6, 46]. The most commonly used cost functions are the *cut-net* metric $\text{cut}(\Pi) := \sum_{e \in E} \omega(e)$ and the *connectivity* metric $(\lambda - 1)(\Pi) := \sum_{e \in E'} (\lambda(e) - 1) \omega(e)$, where E' is the set of all cut nets [23]. In this paper, we use the connectivity-metric, which accurately models the total communication volume of parallel sparse matrix-vector multiplication [16]. Optimizing both objective functions is known to be NP-hard [46]. *Contracting* a pair of vertices (u, v) means merging v into u . The weight of u becomes $c(u) := c(u) + c(v)$. We connect u to the former neighbors $\Gamma(v)$ of v by replacing v with u in all nets $e \in I(v) \setminus I(u)$ and remove v from all nets $e \in I(u) \cap I(v)$. *Uncontracting* a vertex u reverses the contraction. The two most common

ways to represent a hypergraph $H = (V, E, c, \omega)$ as an undirected graph are the *clique* and the *bipartite* representation [38]. In the following, we use *nodes* and *edges* when referring to a graph representation and *vertices* and *nets* when referring to H . In the *clique* graph $G_x(V, E_x \subseteq V^2)$ of H , each net is replaced with an edge for each pair of vertices in the net: $E_x := \{(u, v) : u, v \in e, e \in E\}$. Thus the pins of a net e with size $|e|$ form a $|e|$ -clique in G_x . In the *bipartite* graph $G_*(V \dot{\cup} E, F)$ the vertices and nets of H form the node set and for each net e incident to a vertex v , we add an edge (e, v) to G_* . The edge set F is thus defined as $F := \{(e, v) \mid e \in E, v \in e\}$. Each net in E therefore corresponds to a star in G_* . In both models, node weights c and edge weights ω are chosen according to the problem domain [31].

Related Work. Since the 1990s HGP has evolved into a broad research area. We refer to [6, 9, 53, 58] for an extensive overview, and focus instead on issues closely related to the contributions of our paper. Well-known multilevel HGP software packages with certain distinguishing characteristics include PaToH [16] (originating from scientific computing), hMetis [41, 42] (originating from VLSI design), Mondriaan [61] (sparse matrix partitioning), MLPart [5] (circuit partitioning), Zoltan [22] and Parkway [59] (parallel), UMPa [60] (directed hypergraph model, multi-objective), and kPaToH (multiple constraints, fixed vertices) [8]. All of these algorithms compute vertex matchings [5, 8, 16, 22, 61] or clusterings [16, 41, 42, 59] on each level of the coarsening hierarchy. Different rating functions are used to determine the vertices to be matched or clustered together. All clustering algorithms proceed in a local and greedy fashion: For each vertex the neighbor that maximizes the rating function is chosen as contraction partner. Global decisions are avoided due to the high running times of the respective algorithms [58]. Hagen and Kahng [32] propose a $\mathcal{O}(n^3)$ time algorithm that uses cycles in random walks of the clique representation to identify global clustering structure. Cong and Lim [18] use approximate edge separability computations as a global clustering measure and give an algorithm that runs in $\mathcal{O}(m + n \log n)$ time on the clique representation with m edges and n nodes. Note that for sparse hypergraphs the number of edges in the clique representation can be as high as $m \in \mathcal{O}(n^2)$. Lotfifar and Johnson [47] suggest to cluster hyperedges and to remove less important ones to make better global vertex clustering decisions using rough set clustering.

KaHyPar. The **Karlsruhe Hypergraph Partitioning** framework instantiates the multilevel approach in its most extreme version, removing only a single vertex in every level of the hierarchy. By using this very fine grained n -level approach combined with strong local search heuristics, KaHyPar seems to be the method of choice for optimizing the cut- and the $(\lambda - 1)$ -metric unless speed is more important than quality [1, 56]. Currently, it contains two coarsening algorithms. The first algorithm [56] starts with calculating the locally best contraction partner $u \in \Gamma(v)$ for each vertex v according to a rating function. Then the contractions are performed in decreasing rating score order. Ratings are stored in a priority queue and kept up-to-date during the coarsening process. Thus the algorithm always contracts the vertex pair with the globally highest rating. In the second algorithm vertices are visited in *random* order and each vertex is immediately contracted with its highest-rated neighbor. This approach is shown to not affect the solution quality, while being significantly faster than the first algorithm [1].

Community Detection via Modularity Maximization. Community detection tries to extract an underlying structure from a graph by dividing its nodes into disjoint subgraphs (communities) such that connections are dense *within* subgraphs but sparse between them [28, 55].

Different quality functions are used to judge the goodness of a division into communities. The most popular quality function is the *modularity* of Newman and Girvan [52], which compares the observed fraction of edges within a community with the expected fraction of edges if edges were placed using a random edge distribution that preserves the degree distribution of the graph [27]. More formally, given a graph G and disjoint communities $C = \{C_1, \dots, C_x\}$, modularity is defined as:

$$Q := \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(C_i, C_j) \quad (1)$$

where A_{ij} is the entry of the adjacency matrix A representing edge (i, j) , $m = \frac{1}{2} \sum_{ij} A_{ij}$ is the number of edges in the graph, k_i is the degree of node i , C_i is the community of vertex i , and δ is the Kronecker delta. Note that this can be generalized to weighted graphs: A_{ij} represents the weight of edge (i, j) , $k_i = \sum_j A_{ij}$ is the weighted degree of node i and $m = \frac{1}{2} \sum_{ij} A_{ij}$ is the sum of all edge weights [51]. Modularity optimization is known to be NP-hard [13], but several efficient heuristics exist. A fast and widely used algorithm is the Louvain method introduced by Blondel et al. [12]: Initially, each node is assigned to a community of its own. Then the algorithm proceeds in two phases that are repeated iteratively. In the first phase, nodes are repeatedly assigned to the neighboring community that maximizes the increase in modularity. This local, greedy optimization stops when no further increase is possible. In the second phase, the graph is coarsened according to the community structure discovered in the first phase by contracting each community into a single node. Then, the process starts again on the coarsened graph and is repeated until the maximum modularity is achieved. The communities of the coarsest graph determine the community structure of the input graph. The algorithm has low computational complexity and is thus suitable for large graphs [28, 45]. There exist several definitions of modularity adapted specifically to bipartite graphs [10, 30, 39, 49]. However, we do not consider these in this work, since they do not translate into fast algorithms and therefore only scale to small bipartite graphs [49]. We note that there also exist techniques to detect communities in k -partite, k -uniform hypergraphs. In these approaches, hypergraphs are projected to k bipartite graphs and bipartite modularity measures are used to detect community structures [50].

3 Community-aware Coarsening

There are three main design goals underlying coarsening schemes of multilevel hypergraph partitioning algorithms [42]:

1. Coarsening should successively reduce the size of the nets, because small nets allow move-based local search algorithms to identify moves that improve the solution quality more easily.
2. Coarsening should successively reduce the number of nets in the coarser hypergraphs. This can be accomplished by preferring contractions that create single-vertex nets and leads to simpler instances for initial partitioning, since single-vertex nets cannot be cut.
3. Vertices should be contracted in such a way that the initial partitioning algorithm is able to compute a high-quality solution, i.e. the partition of the coarsest level should not be significantly worse than the final partition of the hypergraph. Therefore, it is necessary that the coarse approximations remain *structurally similar* to the input hypergraph.

To accommodate goals one and two, state-of-the-art HGP libraries use rating functions to identify and contract highly connected vertices such that the number of nets and their size is successively reduced. The most commonly used rating function is *heavy-edge*: Given

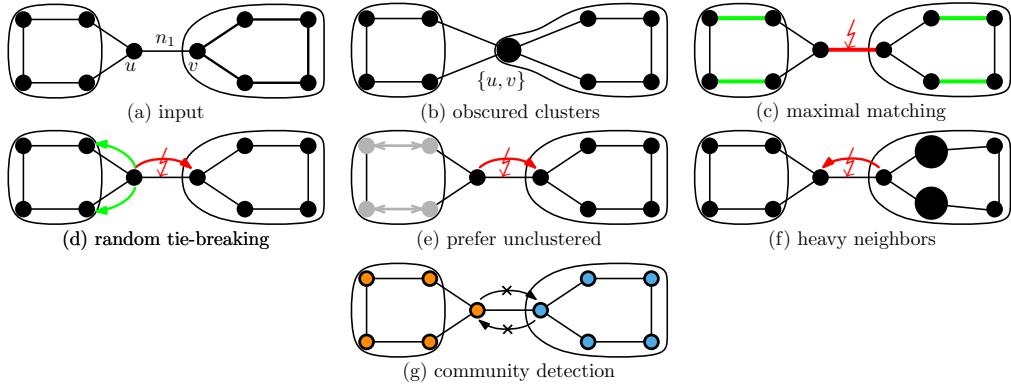


Figure 1 (a) Hypergraph with 10 vertices and 13 nets. Nets containing only two vertices are shown as graph edges. By cutting net n_1 the hypergraph can be partitioned into two balanced blocks. (b) Contracting vertex pair (u, v) obscures the naturally existing clustering structure and the cut of size 1. (c)–(f) Properties of coarsening algorithms that lead to the contraction of (u, v) : (c) Coarsening is based on maximal matchings. (d) Random tie-breaking among all neighbors with same rating score. (e) Preferring unclustered vertices to break ties. (f) Contraction partners with highest rating score are already too heavy. (g) Our approach: Restrict contractions to vertex pairs *within* the same community. This prevents the contraction of (u, v) in all aforementioned cases.

two vertices u and $v \in \Gamma(u)$, it is defined as $r(u, v) := \sum_{e \in E'} \omega(e) / (|e| - 1)$, where $E' := \{I(v) \cap I(u)\}$. This rating is employed in several tools including hMetis [41], Parkway [59], KaHyPar [1, 56] and PaToH [17] and prefers vertex pairs that share a large number of heavy nets with small size. *Structural similarity* between the coarser approximations and the original hypergraph (goal three) is maintained by allowing the formation of vertex clusters instead of enforcing matchings, since their maximality constraint can destroy some naturally existing clusters in the hypergraph [40] (see Fig. 1 (a)–(c) for an example). Furthermore the algorithms ensure that the distribution of vertex weights does not become too imbalanced at the coarsest level, since this limits the number of feasible initial partitions satisfying the balance constraint. This is done by either enforcing an upper-bound on the vertex weight or by integrating a penalty factor into the rating function that discourages the formation of heavy vertices.

However, since coarsening decisions are only based on *local* information, several situations can arise in which naturally existing structure is obscured: If multiple neighbors have the same rating score, coarsening algorithms employ different tie-breaking strategies such as randomly choosing one of them or giving preference to vertices that have not yet been clustered [1, 40] (see Fig. 1 (d),(e)). Furthermore, a restriction on the maximum allowed vertex weight can lead to situations in which the highest rated contractions are forbidden by the weight constraint. Therefore the coarsening algorithm performs a contraction with lower rating score (Figure 1 (f)). Situations like these arise, because all coarsening algorithms are guided by local, greedy decisions based on rating functions that solely consider the weights and sizes of nets connecting candidate vertices and therefore lack a global view of the clustering problem. If information about the community structure were to be known before the coarsening process, these cases could have been prevented explicitly. We therefore propose an approach to combine a *global* view on the problem with *local* coarsening decisions.

Community-aware Coarsening Framework. Our framework consists of two phases. First, a (graph-based) community detection algorithm is used to partition the vertices of the hyper-

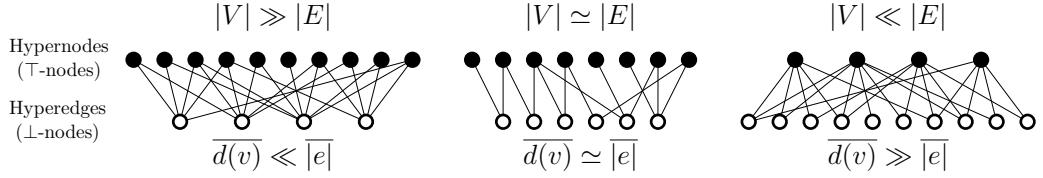


Figure 2 Bipartite graph-based representations of hypergraphs of varying density. In hypergraphs with low density, the bipartite graph consists of many \top -nodes with low average degree and fewer \perp -nodes with high average degree (left). If $d \approx 1$, the number of \top - and \perp -nodes and their average degrees are roughly equal (middle). High-density hypergraphs lead to bipartite representations with fewer \top -nodes with high average degree and many \perp -nodes with low average degree (right).

graph into a set $C = \{C_1, \dots, C_x\}$ of internally densely and externally sparsely connected communities. The actual number of communities $|C|$ is determined by the community detection algorithm. Then, a hypergraph coarsening algorithm is applied on each community C_i independently. This can be accomplished by modifying the algorithm to only contract vertices within the *same* community, i.e. given a vertex $u \in C_i$, we restrict potential contraction partners to $\Gamma(u) \cap C_i$. By preventing inter-community contractions, the coarsening algorithm maintains the structural similarity discovered by the community detection algorithm, while still allowing local, intra-community decisions to be based on rating functions tailored to the HGP problem. Note that this framework is *independent* of the algorithms used for community detection and coarsening. In the following, we describe one instantiation, which performs community detection via modularity maximization using the Louvain method.

Hypergraph Representation. In order to employ the Louvain method as community detection algorithm, a suitable graph-based representation of the hypergraph has to be chosen. As described in Section 2 the two common models are the clique and the bipartite representation. However, several reasons make the clique representation unsuitable for our purpose. Inserting $\binom{|e|}{2}$ graph edges into the clique graph for every net e destroys the natural sparsity of the hypergraph [6] and therefore may be prohibitively costly in terms of both space and running time. Furthermore and more importantly, this exaggerates the importance of nets with more than two pins [57], since large nets automatically imply a high edge density in the clique representation. We therefore use the *bipartite* representation, which allows us to encode any hypergraph in $\mathcal{O}(|P|)$ space. In the following, we refer to the graph nodes representing the vertices of the hypergraph as \top -nodes and to the nodes representing the nets as \perp -nodes (see Figure 2 for an example).

Modeling Peculiarities. By performing community detection on the bipartite graph representation we receive a community partition of *both* the vertices *and* the nets of the hypergraph, since both are represented as (\top, \perp) -nodes in the graph. However, we are only interested in the community structure of the vertices. Therefore we have to take structural properties of the hypergraphs into account. More specifically, we have to consider the density:

$$d := \frac{\overline{d(v)}}{\overline{|e|}} = \frac{|P|/n}{|P|/m} = \frac{m}{n}, \quad (2)$$

where $\overline{d(v)}$ is the average vertex degree and $\overline{|e|}$ is the average net size. If $d \approx 1$, the number of \top -nodes is roughly equal to the number of \perp -nodes and $\overline{d(v)} \approx \overline{|e|}$. If $d \gg 1$ then there are more \perp -nodes than \top -nodes and $\overline{d(v)} \gg \overline{|e|}$, whereas if $d \ll 1$ the opposite is the case (see Figure 2). In case the hypergraph exhibits low density and therefore a large average net size,

special care has to be taken in order to ensure that the community structure is not exclusively shaped by the high-degree \perp -nodes. Similarly, the large number of \perp -nodes can lead to a community structure that is dominated by the nets of the hypergraph in the high-density case. Hypergraphs with density $d \approx 1$ do not pose a problem, since the number of \top -nodes and \perp -nodes as well as their degrees are balanced. We account for these structural differences by encoding additional information about the hypergraph structure into the weights of the bipartite graph edges.

Weighting Graph Edges. We propose three different weights for the edges (v, e) between \top -nodes $v \in V$ and \perp -nodes $e \in E$ as shown in Eq. 3. The first scheme uses uniform edge weights as a baseline. Giving each edge an equal weight is expected to provide good clustering results for hypergraphs with $d \approx 1$, since for these instances the number of \top - and \perp -nodes as well as their degrees are roughly comparable. The second and third schemes account for the skew in low and high density hypergraphs. The weighting function ω_e assigns each edge a weight which is inversely proportional to the size of the net, i.e. smaller nets get a higher influence on the community structure than larger nets. If many small nets are contained within a community, the coarsening algorithm can successively reduce their size and eventually remove them from the hypergraph (goals one and two). Furthermore, this ensures that high-degree \perp -nodes (i.e., large nets) do not dominate the community structure by attracting to many \top -nodes. This edge weight only affects the clustering decisions of \top -nodes, since from the perspective of \perp -nodes each outgoing edge still has uniform weight $1/|e|$. In order to also influence the clustering decision of \perp -nodes, the third weighting function ω_{de} additionally integrates the hypernode degree into the edge weight. Strengthening the connection between \perp -nodes and high-degree \top -nodes facilitates the formation of communities around high-degree vertices in the hypergraph. Note that it is possible to efficiently choose an appropriate weighting scheme at runtime by calculating the density of the hypergraph according to Eq. 2 and modifying the edge weights appropriately.

$$\omega(v, e) := 1 \quad \omega_e(v, e) := \frac{1}{|e|} \quad \omega_{de}(v, e) := \frac{d(v)}{|e|} \quad (3)$$

4 Experimental Evaluation

We implemented modularity-based community detection using the Louvain method in the n -level hypergraph partitioning framework *KaHyPar* (**K**arlsruhe **H**ypergraph **P**artitioning) and modified the default coarsening algorithm [1] to respect the community structure.¹ The code is written in C++ and compiled using g++-5.2 with flags `-O3 -mtune=native -march=native`. We refer to the original algorithm as KaHyPar and to the community-aware versions as *CA*(\cdot), where \cdot is replaced with the appropriate edge weight function. All versions use the default configuration of KaHyPar.

Instances. We evaluate our algorithm on a large collection of 294 hypergraphs [1, 56], which contains instances from three benchmark sets: the ISPD98 VLSI Circuit Benchmark Suite [4], the University of Florida Sparse Matrix Collection [21], and the international SAT Competition 2014 [11]. Sparse Matrices are translated into hypergraphs using the row-net model [16], i.e. each row is treated as a net and each column as a vertex. For

¹ Our implementation is available from <https://github.com/SebastianSchlag/kahypar>.

Table 1 Summary of the hypergraph collection used in the experiments. Instances marked with a * are newly added and were not part of the collection used in [1, 56].

Application	VLSI		Sparse Matrix	SAT Solving			
	ISPD98	DAC2012		SAT14	literal	primal	dual
Benchmark Set			UF-SPM				
Representation	direct	direct	row-net				
Density Class	$d \approx 1$	$d \approx 1$	$d \ll 1, d \approx 1, d \gg 1$	$d \gg 1$	$d \gg 1$	$d \ll 1$	
Community Str.	✓	✓	some instances ³	✓	✓	✓	
# Hypergraphs	18	10*	184	92	92*	92*	
# in Subset	10	5*	60	30	30*	30*	

SAT instances, each boolean *literal* is mapped to one vertex and each clause constitutes a net [53]. In order to incorporate more recent VLSI circuits, we add the instances of the DAC 2012 Routability-Driven Placement Contest [62] to the benchmark set. Furthermore, each SAT instance is also converted into *primal* and *dual* representation [48], which are more common in the SAT solving community than the literal model proposed in [53]. In the *primal* model each variable is represented by a vertex and each clause is represented by a net, whereas in the *dual* model the opposite is the case. While it is known that VLSI circuits and complex networks like web graphs and social networks have a naturally existing clustering structure [25, 27], recent work [7, 29] suggests the same for industrial SAT instances. The complete benchmark set consists of 488 hypergraphs with unit vertex and net weights.² It is used to compare our community-aware algorithm to KaHyPar and to other systems. To study the effects of edge weights on the solution quality for hypergraphs with different densities we use the representative subset of 100 hypergraphs proposed in [56] and add the five smallest DAC2012 hypergraphs as well as the primal and dual representation of each literal SAT hypergraph. In total, the subset therefore consists of 165 hypergraphs, which we divide in three density classes. The class $d \ll 1$ is comprised of all hypergraphs with $d < 0.75$. Hypergraphs with $0.75 \leq d \leq 1.25$ form class $d \approx 1$, while hypergraphs with $d > 1.25$ are assigned to class $d \gg 1$. An overview of our benchmark sets is given in Table 1. While VLSI hypergraphs have $|V| \simeq |E|$ and therefore $d \simeq 1$ [18, 53], SAT hypergraphs exhibit different densities. A primal (or literal) hypergraph of a SAT formula with n variables and $m \in \mathcal{O}(n)$ clauses has density $d \gg 1$, while its dual representation has $d \ll 1$. Instances derived from sparse matrices cover all three cases.

All hypergraphs are partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks with $\varepsilon = 0.03$. For each value of k , a k -way partition is considered to be *one* test instance, resulting in a total of 1155 instances for experiments on the subset and 3416 instances for the full benchmark set.

System and Methodology. All experiments are performed on a single core of a machine consisting of two Intel Xeon E5-2670 Octa-Core processors (Sandy Bridge) clocked at 2.6 GHz. The machine has 64 GB main memory, 20 MB L3- and 8x256 KB L2-Cache and is running RHEL 7.2. To show the effect of community-aware coarsening on the performance of KaHyPar relative to state-of-the-art HGP tools, we compare it with the k -way (hMetis-K) and the recursive bisection variant (hMetis-R) of hMetis 2.0 (p1) [41, 42], and to PaToH 3.2 [16]. These HGP libraries were chosen because they provide the best solution quality [1]. hMetis

² The complete benchmark set along with detailed statistics for each hypergraph is publicly available from <http://algo2.iti.kit.edu/schlag/sea2017/>.

³ Our benchmark set includes hypergraphs derived from web crawls and social networks.

does not directly optimize the $(\lambda - 1)$ metric. Instead it optimizes the *sum-of-external-degrees* (SOED), which is closely related to the connectivity metric: $(\lambda - 1)(\Pi) = \text{SOED}(\Pi) - \text{cut}(\Pi)$ for unweighted hypergraphs (i.e., each cut net contributes λ times its weight to the objective). We therefore set both hMetis versions to optimize SOED and calculate the $(\lambda - 1)$ -metric accordingly. This approach is also used by the authors of hMetis-K [42]. hMetis-R defines the maximum allowed imbalance of a partition differently [41]. An imbalance value of 5, for example, allows each block to weigh between $0.45 \cdot c(V)$ and $0.55 \cdot c(V)$ at each bisection step. We therefore translate our imbalance parameter ε to ε' as described in Eq. (4) such that it matches our balance constraint after $\log_2(k)$ bisections:

$$\varepsilon' := 100 \cdot \left(\left((1 + \varepsilon) \frac{\lceil \frac{c(V)}{k} \rceil}{c(V)} \right)^{\frac{1}{\log_2(k)}} - 0.5 \right). \quad (4)$$

PaToH is configured to use a final imbalance ratio of ε to match our balance constraint. Since PaToH ignores the random seed if configured to use the quality preset, we report both the result of the quality preset (PaToH-Q) and the average over ten repetitions using the default configuration (PaToH-D). All partitioners have a time limit of *eight* hours per test instance. We perform ten repetitions with different seeds for each test instance and report the *arithmetic mean* of the computed cut and running time as well as the best cut found. When averaging over different instances, we use the *geometric mean* in order to give every instance a comparable influence on the final result. In order to compare different algorithms in terms of solution quality, we perform a more detailed analysis using the performance plots introduced in [56]: For each algorithm, these plots relate the smallest minimum cut of all algorithms to the corresponding cut produced by the algorithm on a per-instance basis. For each algorithm, these ratios are sorted in increasing order. The plots use a cube root scale for both axes to reduce right skewness [20] and show $1 - (\text{best}/\text{algorithm})$ on the y-axis to highlight the instances where each partitioner performs badly. A point close to one indicates that the partition produced by the corresponding algorithm was considerably worse than the partition produced by the best algorithm. A value of zero therefore indicates that the corresponding algorithm produced the best solution. Points above one correspond to infeasible solutions that violated the balance constraint. Thus an algorithm is considered to outperform another algorithm if its corresponding ratio values are below those of the other algorithm. In order to include instances with a cut of zero into the results, we set the corresponding cut values to *one* for ratio computations. Furthermore, we conduct Wilcoxon matched pairs signed rank tests [64] (using a 1% significance level) to determine whether or not the difference of KaHyPar-CA and the other algorithms is statistically significant. At a 1% significance level, a Z-score with $|Z| > 2.58$ is considered significant.

Evaluation of Edge Weights. Figure 3 summarizes the results of our experiments on the benchmark subset using different edge weights for the bipartite graph edges. For each density class a box plot shows the improvement of KaHyPar-CA(\cdot) over KaHyPar for initial cuts (computed by the initial partitioning algorithm) and the final average and best cuts (after uncoarsening and local search). Using uniform edge weights for low density hypergraphs worsens the solution quality. However, although the initial cuts are significantly worse in this case, the best cuts are only 2% worse on average than those of KaHyPar. This shows the strength of the n level approach combined with strong local search heuristics. Weighting schemes that encode structural information about the hypergraph into the edge weights perform significantly better. Both CA(ω_e) and CA(ω_{de}) ensure that the community structure of the bipartite graph is not dominated by high-degree \perp -nodes (large nets) by incorporating

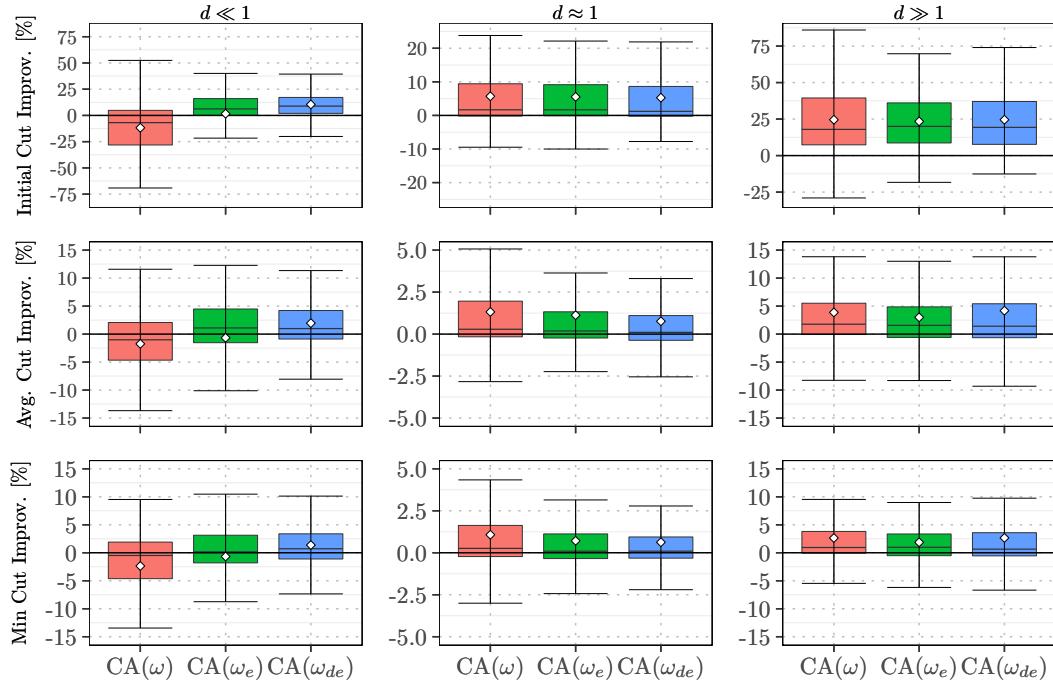


Figure 3 Comparing the improvement of KaHyPar-CA \cdot (using different edge weighting schemes) over KaHyPar on the benchmark subset. Diamonds show the mean improvement.

Table 2 Improvement of KaHyPar-CA over KaHyPar on the benchmark subset. KaHyPar-CA uses $\omega(v, e)$ for hypergraphs with medium and high density and $\omega_{de}(v, e)$ for low-density hypergraphs.

Improvement [%]	VLSI		Sparse Matrix		SAT14		
	DAC2012	ISPD98	All	WebSocial	Primal	Literal	Dual
initial cut	20.5	13.8	4.1	24.8	23.8	34.0	12.2
min cut	3.9	2.0	0.8	3.5	3.5	4.0	1.6
average cut	4.7	2.3	1.1	5.5	4.8	5.7	2.2
worst cut	5.5	2.9	1.5	7.2	6.5	8.0	3.1

the net sizes into the edge weight. However, we can see that CA(ω_{de}) is more stable than CA(ω_e).

Its mean improvement is close to the median, always above zero, and always above the mean improvement of CA(ω_e), which shows that additionally strengthening the connection between \perp -nodes and high-degree \top -nodes indeed has a positive impact on solution quality. For hypergraphs with density $d \approx 1$ uniform edge weights perform best. If the density of the hypergraph is large, all three schemes give comparable results. This can be explained by the fact that if $d \gg 1$, most nets are small. This translates to “small stars” in the bipartite graph (or even paths for nets with $|e| = 2$), which do not distort the community structure of \top -nodes. Based on these results, we configure the final version of our algorithm to choose the weighting scheme at runtime depending on the observed density. If $d \geq 0.75$, it uses uniform edge weights, otherwise $\omega_{de}(v, e)$.⁴ In the following we will refer to this configuration as KaHyPar-CA. As can be seen in Table 2 KaHyPar-CA significantly improves the initial

⁴ Figure 5 in Appendix A compares all three edge weighting schemes and validates this decision.

■ **Table 3** Comparing the average running times of KaHyPar-CA with KaHyPar and other tools.

Algorithm	Running Time [s]							
	All	DAC2012	ISPD98	Primal	Literal	Dual	SPM	WebSocial
KaHyPar	20.4	289.5	8.1	15.6	30.6	57.8	10.9	66.7
KaHyPar-CA	31.0	369.0	12.3	32.9	64.7	68.3	13.9	67.1
hMetis-R	79.2	446.4	29.0	66.2	142.1	200.4	41.8	89.7
hMetis-K	57.9	240.9	23.2	44.2	94.9	125.6	36.0	111.9
PaToH-Q	5.9	28.3	1.9	6.9	9.2	10.6	3.4	4.7
PaToH-D	1.2	6.5	0.4	1.1	1.6	2.9	0.8	0.9

cuts on all benchmark sets. The improvements in average cut (up to 5.7%) and min-cut (up to 4.0%) indicate that KaHyPar-CA is indeed able to compute better solutions than KaHyPar. Furthermore, the fact that the worst solutions of KaHyPar-CA are significantly better (up to 8.0%) than those of KaHyPar shows that community-aware coarsening improves the partitioner’s robustness.

Comparison with other Systems. In the following, we exclude 194 out of 3416 instances because either PaToH-Q could not allocate enough memory or other partitioners did not finish in time. Excluded instances are shown in Appendix B. The following comparison is therefore based on the remaining 3222 instances.⁵ As can be seen in Figure 4 and Table 4, KaHyPar-CA performs significantly better than KaHyPar on *all* benchmark sets. Looking at the solution quality of all systems across all instances (top left), KaHyPar-CA produced the best partitions for 1346 of the 3222 instances. It is followed by hMetis-R (882), KaHyPar (734) and hMetis-K (460). PaToH-D and PaToH-Q computed the best partitions for 163 instances. Note that for some instances multiple partitioners computed the same solution. Comparing the best solutions of KaHyPar-CA to each partitioner individually, KaHyPar-CA produced better partitions than PaToH-D, PaToH-Q, hMetis-K, KaHyPar, hMetis-R in 2849, 2833, 2084, 1979, 1937 cases, respectively.

By using community-aware coarsening, KaHyPar-CA performs best on each of the benchmark sets. As can be seen in Table 4, the difference in solution quality is statistically significant for all benchmark sets except DAC2012, where KaHyPar-CA is on par with hMetis-R. For hypergraphs derived from matrices of web graphs and social networks⁶, KaHyPar-CA dominates all other systems by computing the best partitions for 86 of the 115 instances. Table 3 compares the running times of all partitioners. Although community detection using the Louvain method is itself a multilevel algorithm executed on the bipartite graph representation, KaHyPar-CA remains on average faster than hMetis.

5 Conclusions and Future Work

We describe an improved coarsening scheme for hypergraph partitioning that incorporates *global* information about the structure of the hypergraph by detecting communities in the

⁵ Interactive visualizations of the performance plots and detailed per-instance results can be found on the website accompanying this publication: <http://algo2.iti.kit.edu/schlag/sea2017/>.

⁶ Based on the following matrices: webbase-1M, ca-CondMat, soc-sign-epinions, wb-edu, IMDB, as-22july06, as-caida, astro-ph, HEP-th, Oregon-1, Reuters911, PGgiantcompo, NotreDame-www, NotreDame_actors, p2p-Gnutella25, Stanford, cnr-2000.

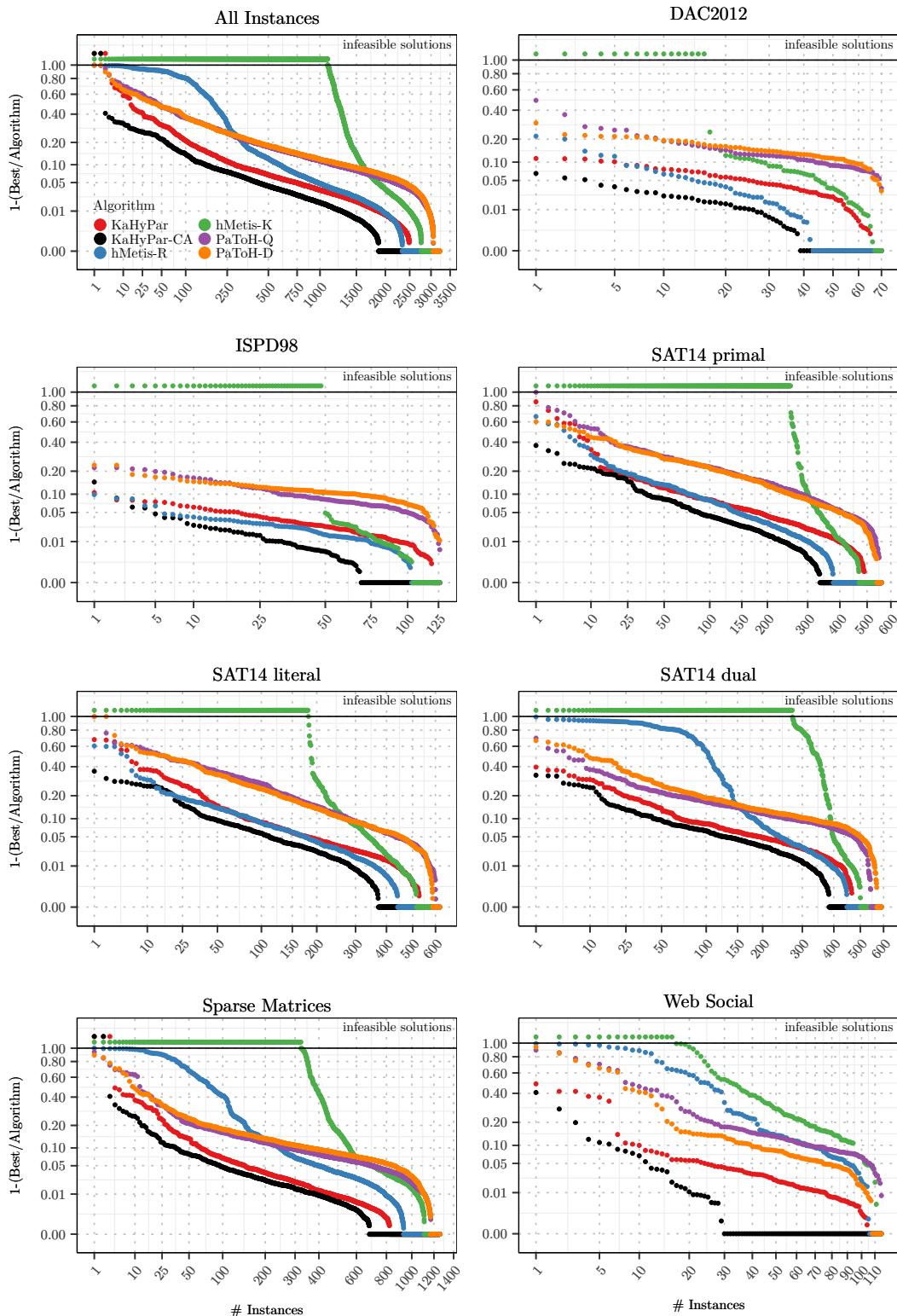


Figure 4 Min-Cut performance plots comparing KaHyPar-CA with KaHyPar and other systems. The y-axis shows the ratio between the smallest cut of all algorithms and the cut produced by the corresponding algorithm.

Table 4 Results of significance tests comparing KaHyPar-CA with KaHyPar and other systems on the full benchmark set. We report the Z -scores and p -values of the Wilcoxon matched pairs signed rank tests. At a 1% significance level, a Z -score with $|Z| > 2.58$ is considered significant. Negative Z -scores hereby indicate that KaHyPar-CA performs better than the respective algorithm. Note that hMetis-K has slight advantages in the following comparisons because we do not disqualify imbalanced partitions in the statistical analysis.

Class	Algorithm	KaHyPar-CA	
		Z	p
DAC2012	KaHyPar	−6.168	6.907e-10
	hMetis-R	−1.484	0.1379
	hMetis-K	−6.487	8.748e-11
	PaToH-D	−7.271	3.559e-13
	PaToH-Q	−7.271	3.559e-13
ISPD98	KaHyPar	−7.962	1.695e-15
	hMetis-R	−5.806	6.403e-09
	hMetis-K	−2.751	0.005935
	PaToH-D	−9.638	5.522e-22
	PaToH-Q	−9.636	5.655e-22
SAT14 Primal	KaHyPar	−11.22	3.232e-29
	hMetis-R	−4.411	1.027e-05
	hMetis-K	−6.918	4.579e-12
	PaToH-D	−17.23	1.56e-66
	PaToH-Q	−17.69	5.403e-70
SAT14 Literal	KaHyPar	−11.3	1.354e-29
	hMetis-R	−4.189	2.802e-05
	hMetis-K	−5.475	4.375e-08
	PaToH-D	−19.33	3.162e-83
	PaToH-Q	−19.56	3.12e-85
SAT14 Dual	KaHyPar	−7.271	3.573e-13
	hMetis-R	−8.339	7.515e-17
	hMetis-K	−8.071	6.969e-16
	PaToH-D	−18.21	4.656e-74
	PaToH-Q	−16.04	6.727e-58
UF-SPM	KaHyPar	−5.941	2.832e-09
	hMetis-R	−16.75	5.467e-63
	hMetis-K	−21.81	1.739e-105
	PaToH-D	−26.83	1.557e-158
	PaToH-Q	−25.39	3.612e-142
WebSocial	KaHyPar	−7.164	7.839e-13
	hMetis-R	−8.776	1.7e-18
	hMetis-K	−9.151	5.647e-20
	PaToH-D	−8.721	2.755e-18
	PaToH-Q	−7.368	1.737e-13

bipartite graph representation via modularity maximization using the Louvain method. We make this approach suitable for a wide spectrum of instances by appropriately choosing weights for the graph edges based on the density of the hypergraph. Experiments on a large benchmark set demonstrate that community-aware coarsening significantly improves the partitioning quality of KaHyPar on *all* instance classes, while having only a moderate impact on the overall running time. On all but one class, KaHyPar-CA performs statistically significantly better than KaHyPar, hMetis, and PaToH and is on par with the best partitioner otherwise.

There exist several ideas for future work. Given the significantly improved initial cuts, it might be feasible to equip KaHyPar with faster (and less strong) local search algorithms to narrow the gap between the running time of KaHyPar and PaToH. Modularity maximization is widely used to detect community structure but also known to exhibit a certain scaling behavior and resolution limit [26]. Future work therefore includes the analysis of whether these limitations negatively affect the coarsening process and if multi-resolution modularity [44] can be used as a remedy. Furthermore, there exist several alternative approaches to community detection such as Infomap [54] and Surprise [2] that could also be evaluated in our community-aware coarsening framework.

References

- 1 Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag. Engineering a direct k -way hypergraph partitioning algorithm. In *19th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 28–42, 2017.
- 2 Rodrigo Aldecoa and Ignacio Marin. Deciphering Network Community Structure by Surprise. *PLOS ONE*, 6(9):1–8, 09 2011.
- 3 F. A. Aloul, I. L. Markov, and K. A. Sakallah. MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation. *Journal of Universal Computer Science*, 10(12):1562–1596, 2004.
- 4 C. J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proceedings of the 1998 International Symposium on Physical Design*, pages 80–85. ACM, 1998.
- 5 C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel Circuit Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, 1998.
- 6 C. J. Alpert and A. B. Kahng. Recent Directions in Netlist Partitioning: a Survey. *Integration, the VLSI Journal*, 19(1–2):1 – 81, 1995.
- 7 C. Ansótegui, J. Giráldez-Cru, and J. Levy. The community structure of sat formulas. In Alessandro Cimatti and Roberto Sebastiani, editors, *15th International Conference of Theory and Applications of Satisfiability Testing (SAT)*, pages 410–423. Springer, 2012.
- 8 C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level Direct K-way Hypergraph Partitioning with Multiple Constraints and Fixed Vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, 2008.
- 9 D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Proc. Graph Partitioning and Graph Clustering – 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*. AMS, 2013.
- 10 M. J. Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76:066102, Dec 2007.
- 11 A. Belov, D. Diepold, M. Heule, and M. Järvisalo. The SAT Competition 2014. <http://www.satcompetition.org/2014/>, 2014.
- 12 V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

- 13 U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. *IEEE Trans. Knowledge and Data Engineering*, 20(2):172–188, 2008.
- 14 T. N. Bui and C. Jones. Finding Good Approximate Vertex and Edge Partitions is NP-Hard. *Information Processing Letters*, 42(3):153–159, 1992.
- 15 T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452, 1993.
- 16 Ü. V. Catalyürek and C. Aykanat. Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, Jul 1999.
- 17 Ü. V. Catalyürek and C. Aykanat. PaToH: Partitioning Tool for Hypergraphs. <http://bmi.osu.edu/umit/PaToH/manual.pdf>, 1999.
- 18 J. Cong and S. K. Lim. Edge separability-based circuit clustering with application to multilevel circuit partitioning. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(3):346–357, 2004.
- 19 J. Cong and M. Smith. A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design. In *30th Conference on Design Automation*, pages 755–760, June 1993.
- 20 N. J. Cox. Stata tip 96: Cube roots. *Stata Journal*, 11(1):149–154(6), 2011. URL: <http://www.stata-journal.com/article.html?article=st0223>.
- 21 T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011.
- 22 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 Conference on Parallel and Distributed Processing*, IPDPS, pages 124–124. IEEE, 2006.
- 23 W. E. Donath. Logic partitioning. *Physical Design Automation of VLSI Systems*, pages 65–86, 1988.
- 24 V. Durairaj and P. Kalla. Guiding CNF-SAT Search via Efficient Constraint Partitioning. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided Design*, ICCAD, pages 498–501. IEEE, 2004.
- 25 S. Dutt and W. Deng. VLSI Circuit Partitioning by Cluster-removal Using Iterative Improvement Techniques. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, ICCAD, pages 194–200. IEEE, 1996.
- 26 S. Fortunato and M. Barthélémy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.
- 27 S. Fortunato and D. Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.
- 28 Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.
- 29 J. Giráldez-Cru and J. Levy. Generating sat instances with community structure. *Artificial Intelligence*, 238:119 – 134, 2016.
- 30 R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76:036102, Sep 2007.
- 31 S. W. Hadley. Approximation Techniques for Hypergraph Partitioning Problems. *Discrete Applied Mathematics*, 59(2):115–127, 1995.
- 32 L. Hagen and A. B. Kahng. A New Approach to Effective Circuit Clustering. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-aided Design*, ICCAD, pages 422–427. IEEE, 1992.
- 33 S. Hauck and G. Borriello. An Evaluation of Bipartitioning Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):849–866, Aug 1997.

- 34 B. Heintz and A. Chandra. Beyond graphs: Toward scalable hypergraph analysis systems. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):94–97, April 2014.
- 35 B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *International Symposium on Solving Irregularly Structured Problems in Parallel*, pages 218–225, 1998.
- 36 B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000.
- 37 B. Hendrickson and R. Leland. A Multi-Level Algorithm For Partitioning Graphs. *SC Conference*, 0:28, 1995.
- 38 T. C. Hu and K. Moerder. Multiterminal Flows in a Hypergraph. In T.C. Hu and E.S. Kuh, editors, *VLSI Circuit Layout: Theory and Design*, chapter 3, pages 87–93. IEEE Press, 1985.
- 39 K. Suzuki and K. Wakita. Extracting Multi-facet Community Structure from Bipartite Networks. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE*, pages 312–319, 2009.
- 40 G. Karypis. Multilevel hypergraph partitioning. In Jason Cong and Joseph R. Shinnerl, editors, *Multilevel Optimization in VLSICAD*, pages 125–154. Springer, 2003.
- 41 G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on Very Large Scale Integration VLSI Systems*, 7(1):69–79, 1999.
- 42 G. Karypis and V. Kumar. Multilevel K -way Hypergraph Partitioning. In *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pages 343–348. ACM, 1999.
- 43 S. Klamt, U. Haus, and F. Theis. Hypergraphs and Cellular Networks. *PLOS Computational Biology*, 5(5):e1000385, 05 2009.
- 44 R. Lambiotte. Multi-scale modularity in complex networks. In *8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 546–553, May 2010.
- 45 A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80:056117, Nov 2009.
- 46 T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., 1990.
- 47 F. Lotififar and M. Johnson. A Multi-level Hypergraph Partitioning Algorithm Using Rough Set Clustering. In *21st International Conference on Parallel and Distributed Computing (Euro-Par)*, pages 159–170, 2015.
- 48 Z. Mann and P. Papp. Formula partitioning revisited. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 41–56. EasyChair, 2014.
- 49 T. Murata. Modularity for Bipartite Networks. In N. Memon, J. J. Xu, D. L. Hicks, and H. Chen, editors, *Data Mining for Social Network Data*. Springer, 2010.
- 50 N. Neubauer and K. Obermayer. Towards community detection in k-partite k-uniform hypergraphs. In *Proceedings of the NIPS 2009 Workshop on Analyzing Networks and Learning with Graphs*, pages 1–9, 2009.
- 51 M. E. J. Newman. Analysis of weighted networks. *Physical Review E*, 70:056131, Nov 2004.
- 52 M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, Feb 2004.
- 53 D. A. Papa and I. L. Markov. Hypergraph Partitioning and Clustering. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.
- 54 M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. *The European Physical Journal Special Topics*, 178(1):13–23, 2009.

- 55 S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, August 2007.
- 56 S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. *k*-way Hypergraph Partitioning via n -Level Recursive Bisection. In *18th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67, 2016.
- 57 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, pages 57–62. ACM, 1972.
- 58 A. Trifunovic. *Parallel Algorithms for Hypergraph Partitioning*. PhD thesis, University of London, 2006.
- 59 A. Trifunović and W. J. Knottenbelt. Parallel Multilevel Algorithms for Hypergraph Partitioning. *Journal of Parallel and Distributed Computing*, 68(5):563 – 581, 2008.
- 60 Ü. V. Çatalyürek and M. Deveci and K. Kaya and B. Uçar. UMPA: A multi-objective, multi-level partitioner for communication minimization. In Bader et al. [9], pages 53–66.
- 61 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.
- 62 N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. The DAC 2012 Routability-driven Placement Contest and Benchmark Suite. In *Proceedings of the 49th Annual Design Automation Conference*, DAC’12, pages 774–782. ACM, 2012.
- 63 S. Wichlund. On multilevel circuit partitioning. In *1998 International Conference on Computer-aided Design*, ICCAD, pages 505–511. ACM, 1998.
- 64 F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945. URL: <http://www.jstor.org/stable/3001968>.

A Performance Plots of Edge Weighting Schemes

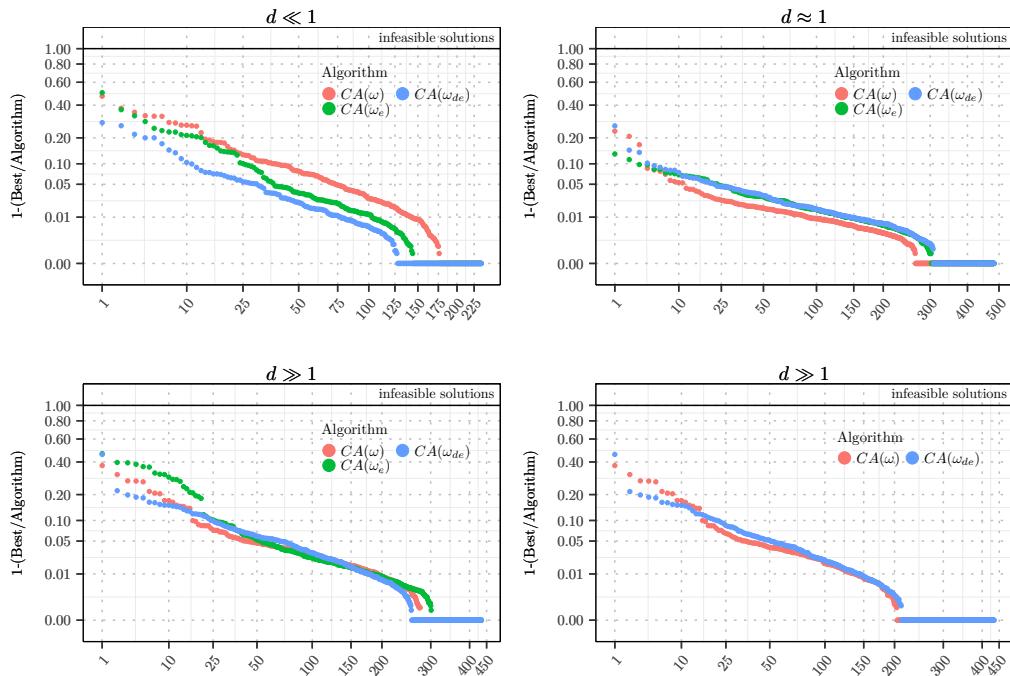


Figure 5 Min-Cut performance plots comparing the different edge weighting schemes of KaHyPar-CA on the benchmark subset.

B Excluded Test Instances

Out of 3416 test instances, we excluded the following 194 instances either because PaToH-Q could not allocate enough memory or one of the other partitioners could not partition the instances in the given time limit. The table is split into two groups: SAT instances and sparse matrix instances. Note that whenever KaHyPar or KaHyPar-CA exceeded the time limit, it was due to the long running time of local search.

Table 5 Instances excluded from the full benchmark set evaluation.

Hypergraph	2	4	8	16	32	64	128
10pipe-q0-k.dual				△	△	△	○△
10pipe-q0-k.primal	□	□	□	□	□	□	□
11pipe-k.dual	△	○△	○△	○△	○△	○△	○△
11pipe-k				○	○	○	○
11pipe-k.primal	□	□	□	□	□	□	○□
11pipe-q0-k.dual					△	○△	○△
11pipe-q0-k.primal	□	□	□	□	□	□	□
9dlx-vliw-at-b-iq3.dual							△
9dlx-vliw-at-b-iq3.primal	□	□	□	□	□	□	□
9vliw-m-9stages-iq3-C1-bug7.dual	△	●○△	●○△	●○△	●○△	●○△	●○△
9vliw-m-9stages-iq3-C1-bug7	△	△	●○△	●○△	●○△	●○□△	●○□△
Hypergraph	2	4	8	16	32	64	128
9vliw-m-9stages-iq3-C1-bug7.primal	△	△		△	○△	○△	○△
9vliw-m-9stages-iq3-C1-bug8.dual	△	●○△	●○△	●○△	●○△	●○△	●○△
9vliw-m-9stages-iq3-C1-bug8	△	△	●○△	●○△	●○△	●○□△	●○□△
9vliw-m-9stages-iq3-C1-bug8.primal	△	△		△	○△	○△	○△
blocks-blocks-37-1.130-NOTKNOWN.dual	○	●○	●○	●○	●○	●○	●○△
blocks-blocks-37-1.130-NOTKNOWN	□	□	□	□	□	□	□
blocks-blocks-37-1.130-NOTKNOWN.primal	□	□	□	□	□	□	□
E02F20.dual							○
E02F22.dual						○	○
openstacks-p30-3.085-SAT.primal	□	□	□	□	□	□	□
openstacks-sequencedstrips-nonadl-nonnegated-os-sequencedstrips-p30-3.025-NOTKNOWN.primal	□	□	□	□	□	□	□
openstacks-sequencedstrips-nonadl-nonnegated-os-sequencedstrips-p30-3.085-SAT.primal	□	□	□	□	□	□	□
transport-transport-city-sequential-25nodes-1000size-3degree-100mindistance-3trucks-10packages-2008seed.030-NOTKNOWN.dual							△

transport-transport-city-sequential-25nodes-1000size-3degree-100mindistance-3trucks-10packages-2008seed.050-NOTKNOWN.primal	□	□	□	□	
q-query-3-L100-coli.sat.dual					△
q-query-3-L150-coli.sat.dual				△	△
q-query-3-L200-coli.sat.dual			△	△	△
q-query-3-L80-coli.sat.dual					△
velev-vliw-uns-2.0-uq5.dual		△	△	△	△
velev-vliw-uns-2.0-uq5.primal	□	□	□	□	□
velev-vliw-uns-4.0-9.dual				△	△
velev-vliw-uns-4.0-9.primal	□	□	□	□	□
192bit	□		□		
appu				○	○
ESOC	□	□		□	○□
human-gene2				○	○
IMDB			△	△	△
on-g500-logn16	△	△	△	△	○△
Rucci1				□	
sls	□	□	□	○□	○□
Trec14					○

- △ : KaHyPar/KaHyPar-CA exceeded time limit
- : hMetis-R exceeded time limit
- : hMetis-K exceeded time limit
- : PaToH-Q memory allocation error