Network Flow-Based Refinement for Multilevel Hypergraph Partitioning

TOBIAS HEUER, PETER SANDERS, and SEBASTIAN SCHLAG, Karlsruher Institut für Technologie (KIT), Germany

We present a refinement framework for multilevel hypergraph partitioning that uses max-flow computations on pairs of blocks to improve the solution quality of a k-way partition. The framework generalizes the flow-based improvement algorithm of the Karlsruhe Fast Flow Partitioner (KaFFPa) from graphs to hypergraphs and is integrated into the hypergraph partitioner Karlsruhe Hypergraph Partitioning (KaHyPar). By reducing the size of hypergraph flow networks, improving the flow model used in KaFFPa, and developing techniques to improve the running time of our algorithm, we obtain a partitioner that computes the best solutions for a wide range of benchmark hypergraphs from different application areas for both the connectivity and the cut-net metric while still having a running time comparable to that of hMetis. In the case of graph partitioning, our algorithm compares favorably with KaFFPa, even after enhancing the latter with our improved flow network, and at the same time is more than a factor of two faster. Finally, we show that our algorithm improves the performance of the memetic multilevel hypergraph partitioner KaHyPar-E.

CCS Concepts: • Mathematics of computing → Hypergraphs; Network flows; Graph algorithms;

Additional Key Words and Phrases: Multilevel hypergraph partitioning, network flows, refinement

ACM Reference format:
Tobias Heuer, Peter Sanders, and Sebastian Schlag. 2019. Network Flow-Based Refinement for Multilevel Hypergraph Partitioning. J. Exp. Algorithmics 24, 2, Article 2.3 (September 2019), 36 pages.
https://doi.org/10.1145/3329872

1 INTRODUCTION

Hypergraphs are a generalization of graphs, where each (hyper)edge (or 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 + ε times the average block size), while minimizing an objective function over the nets. However, allowing nets of arbitrary size makes the partitioning problem more difficult in practice [21, 41]. The two most prominent objective functions are the cut-net and the connectivity (or λ − 1) metrics. Cut-net is a straightforward generalization of the edge-cut objective in graph partitioning (i.e., minimizing the sum of the weights of those nets that connect more than one block).
The connectivity metric additionally takes into account the actual number $\lambda$ of blocks connected by a net. By summing the $(\lambda - 1)$-values of all nets, one accurately models the total communication volume of parallel sparse matrix-vector multiplication \cite{16} and once more gets a metric that reverts to edge-cut for plain graphs.

It is well known that hypergraph partitioning (HGP) is NP-hard \cite{56}, which is why practical applications mostly use heuristic multilevel algorithms \cite{13,19,40,42}. These algorithms successively contract the hypergraph to obtain a hierarchy of smaller, structurally similar hypergraphs. After applying an initial partitioning algorithm to the smallest hypergraph, contraction is undone and, at each level, a local search method is used to improve the partitioning induced by the coarser level. A $k$-way partition is either computed directly or via recursive bisection. In direct $k$-way partitioning, the hypergraph is first coarsened and afterwards, an initial partitioning algorithm computes a $k$-way partition. This partition is then refined during uncoarsening using $k$-way local search algorithms. Recursive bisection (RB) algorithms on the other hand obtain the final $k$-way partition by first computing a bisection of the initial hypergraph using the multilevel approach and then recursing on each of the two blocks.

All state-of-the-art HGP algorithms \cite{2,4,7,25,46,50,52,69,74,75,77} either use simple greedy algorithms \cite{51,52}, or variations of the Kernighan-Lin (KL) algorithm \cite{53,70} or the Fiduccia-Mattheyses (FM) heuristic \cite{29,66} for local search. These heuristics move vertices between blocks in descending order of improvements in the optimization objective (gain) and are known to be prone to get stuck in local optima when used directly on the input hypergraph \cite{52}. The multilevel paradigm helps to some extent, since it allows a more global view on the problem on the coarse levels and a very fine-grained view on the fine levels of the hierarchy. However, the performance of move-based approaches degrades for hypergraphs with large hyperedges. In these cases, it is difficult to find meaningful vertex moves that improve the solution quality because large hyperedges are likely to have many vertices in multiple blocks \cite{76}. Thus, the gain of moving a single vertex to another block is likely to be zero \cite{59}.

While finding balanced minimum cuts in hypergraphs is NP-hard, a minimum cut separating two vertices can be found in polynomial time using network flow algorithms and the max-flow min-cut theorem \cite{32}. Flow algorithms find an optimal min-cut and do not suffer the drawbacks of move-based approaches. However, they were long overlooked as heuristics for balanced partitioning due to their high complexity \cite{58,80}. Sanders and Schulz \cite{67} presented a max-flow-based improvement algorithm for graph partitioning, which is integrated into the multilevel Karlsruhe Fast Flow Partitioner (KaFFPa) and computes high-quality solutions.

Outline and Contribution. Motivated by the results of Sanders and Schulz \cite{67}, we generalize the max-flow min-cut refinement framework of KaFFPa from graphs to hypergraphs. Section 2 first introduces basic notation and gives a brief overview of related work. In Section 3, we then review the KaFFPa approach and propose a simple modification to overcome shortcomings of its flow network, which restrict the search space of feasible solutions. Section 4.1 explains how hypergraphs are transformed into flow networks and presents a technique to reduce the size of the resulting hypergraph flow network. Section 4.2 shows how this network can be used to construct a flow problem such that the min-cut induced by a max-flow computation between a pair of blocks improves either the cut-net or connectivity objective of a $k$-way partition. We furthermore show how the modification of KaFFPa’s flow network can be generalized to hypergraphs by exploiting the structure of hypergraph flow networks. We implemented our algorithm in the open source HGP framework Karlsruhe Hypergraph Partitioning (KaHyPar) and, therefore, briefly discuss implementation details and techniques to improve the running time in Section 4.3.

Extensive experiments presented in Section 5 demonstrate that our flow model yields better solutions than the KaFFPa approach for both hypergraphs and graphs. We furthermore show that
using pairwise flow-based refinements significantly improves partitioning quality by augmenting the previously best direct $k$-way and recursive bisection-based configurations of KaHyPar with our algorithm. For connectivity optimization, the resulting direct $k$-way hypergraph partitioner KaHyPar-MF performs better than all competing algorithms on all instance classes and still has a running time comparable to that of hMetis. It furthermore performs well when optimizing the cut-net metric. Only for bipartitioning and a benchmark set containing hypergraphs with many large nets, KaHyPar-R-MF, a recursive bisection-based configuration augmented with flow-based refinements, is able to compute solutions of comparable quality. When used as a graph partitioner, KaHyPar-MF compares favorably with KaFFPa in terms of solution quality, even when the latter is enhanced with our improved flow model described in Section 3.2, while at the same time being more than a factor of two faster. Finally we show that our new algorithm is useful as partitioning engine in the memetic hypergraph partitioner KaHyPar-E [6]. Earlier versions of our work have been uploaded to ArXiv [44] and presented at the SEA 2018 conference [45].

2 PRELIMINARIES

2.1 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 $e$ is a subset of the vertex set $V$ (i.e., $e \subseteq V$). The vertices of a net are called pins. We extend $c$ and $\omega$ 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 size $|e|$ of a net $e$ is the number of its pins. Given a subset $V' \subseteq V$, the subhypergraph $H_{V'}$ is defined as $H_{V'} := (V', \{v \cap V' \mid v \in E : e \cap V' \neq \emptyset\})$.

A $k$-way partition of a hypergraph $H$ is a partition of its vertex set into $k$ blocks $\Pi = \{V_1, \ldots, V_k\}$ 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 $\Pi$ $\epsilon$-balanced if each block $V_i \in \Pi$ satisfies the balance constraint: $c(V_i) \leq L_{\text{max}} := (1 + \epsilon)|E|/k$ for some parameter $\epsilon$. For each net $e$, $\Lambda(e) := \{V_i \mid V_i \cap e \neq \emptyset\}$ denotes the connectivity set of $e$. The connectivity of a net $e$ is $\lambda(e) := |\Lambda(e)|$. A net is called cut net if $\lambda(e) > 1$. A vertex $u$ incident to at least one cut net is called a boundary vertex. The number of pins of a net $e$ in block $V_j$ is defined as $\Phi(e, V_j) := \{|e \mid v \in V_j \mid v \in \epsilon\}$. Given a $k$-way partition $\Pi$ of $H$, the quotient graph $Q := (\Pi, \{(V_i, V_j) \mid \exists e \in E : (V_i, V_j) \subseteq \Lambda(e)\})$ contains an edge between each pair of adjacent blocks.

The $k$-way hypergraph partitioning problem is to find an $\epsilon$-balanced $k$-way partition $\Pi$ of a hypergraph $H$ that minimizes an objective function over the cut nets for some $\epsilon$. The most commonly used cost functions are the cut-net metric $\text{cut}(\Pi) := \sum_{e \in E'} \omega(e)$ and the connectivity metric $\lambda^{-1}(\Pi) := \sum_{e \in E'} (\lambda(e) - 1)\omega(e)$ [1], where $E'$ is the set of all cut nets [27]. Optimizing each of the objective functions is known to be NP-hard [56]. Note that both cost functions revert to edge-cut for plain graphs. In the following, we use vertices and nets when referring to hypergraphs and nodes and edges when referring to graphs. Furthermore, we treat the graph partitioning problem minimizing the edge cut as a special case of the hypergraph partitioning problem.

The star-expansion [48] of a hypergraph $H$ is the bipartite graph $G_\ast(V \cup E, F)$ in which the vertices and nets of $H$ form the node set and for each net $e \in I(v)$, an edge $(e, v)$ is added to $F$. Each net in $E$ thus corresponds to a star in $G_\ast$.

Let $G = (V, E, c, \omega)$ be a weighted (directed) graph. We use the same notation to refer to node weights $c$, edge weights $\omega$, and node degrees $d(v)$. Furthermore, $\Gamma(u) := \{v : (u, v) \in E\}$ denotes the neighbors of node $u$. In an undirected graph, an edge $(u, v) \in E$ implies an edge $(v, u) \in E$ and $\omega(u, v) = \omega(v, u)$. A path $P = (v_1, \ldots, v_k)$ is a sequence of nodes such that each pair of consecutive nodes is connected by a directed edge. A strongly connected component $C \subseteq V$ is a set of nodes such
that for each \( u, v \in C \) there exists a path from \( u \) to \( v \). A topological ordering is a linear ordering \( \prec \) of \( V \) such that every directed edge \((u, v) \in E\) implies \( u \prec v \) in the ordering. A set of nodes \( B \subseteq V \) is called a closed set iff there are no outgoing edges leaving \( B \), i.e., if the conditions \( u \in B \) and \((u, v) \in E\) imply \( v \notin B \). A subset \( S \subseteq V \) is called a node separator if its removal divides \( G \) into two disconnected components. Given a subset \( V' \subset V \), the induced subgraph \( G[V'] \) is defined as \( G[V'] := (V', \{(u, v) \in E \mid u, v \in V'\}) \).

A flow network \( N = (V, \mathcal{E}, c) \) is a directed graph with two distinguished nodes \( s \) and \( t \) in which each edge \( e \in \mathcal{E} \) has a capacity \( c(e) \geq 0 \). An \((s, t)\)-flow (or flow) is a function \( f : V \times V \to \mathbb{R} \) that satisfies the capacity constraint \( \forall u, v \in V : f(u, v) \leq c(u, v) \), the skew symmetry constraint \( \forall u, v \in V \times V : f(u, v) = -f(v, u) \), and the flow conservation constraint \( \forall v \in V \setminus \{s, t\} : \sum_{u \in V} f(u, v) = 0 \). The value of a flow \(|f| := \sum_{u \in V} f(s, u)\) is defined as the total amount of flow transferred from \( s \) to \( t \). The residual capacity is defined as \( r_f(u, v) = c(u, v) - f(u, v) \). An edge \( e \) is called saturated iff \( r_f(e) = 0 \). Given a flow \( f \), \( \mathcal{N}_f = (\mathcal{V}, \mathcal{E}_f, r_f) \) with \( \mathcal{E}_f = \{(u, v) \in \mathcal{V} \times \mathcal{V} \mid r_f(u, v) > 0\} \) is the residual network. An \((s, t)\)-cut (or cut) is a bipartition \((\mathcal{S}, \mathcal{V} \setminus \mathcal{S})\) of a flow network \( N \) with \( s \in \mathcal{S} \subset \mathcal{V} \) and \( t \in \mathcal{V} \setminus \mathcal{S} \). The capacity of an \((s, t)\)-cut is defined as \( \sum_{e \in \mathcal{E}} c(e) \), where \( \mathcal{E}' = \{(u, v) \in \mathcal{E} : u \in \mathcal{S}, v \in \mathcal{V} \setminus \mathcal{S}\} \). The max-flow min-cut theorem states that the value \(|f|\) of a maximum flow \( f \) is equal to the capacity of a minimum cut separating \( s \) and \( t \) [32].

### 2.2 Related Work

Hypergraph Partitioning. HGP has evolved into a broad research area since the 1990s. We refer to existing literature [5, 8, 62, 73] for an extensive overview. Well-known sequential multilevel systems with certain distinguishing characteristics include PaToH [16] (originating from scientific computing), hMetis [51, 52] (originating from VLSI design), KaHyPar [2, 46, 69] (n-level, general purpose), Mondriaan [77] (targeted at partitioning sparse rectangular matrices), MLPart [4] (targeted at circuit partitioning), Zoltan-AlgD [71] (coarsening inspired by algebraic multigrid methods), UMPa [75] (directed hypergraph model, multi-objective partitioning), and kPaToH [7] (multiple constraints, fixed vertices). Distributed HGP systems include Zoltan [25] and Parkway [74] (multilevel), and SHP [50] (non-multilevel). All multilevel tools use move-based local search algorithms in the refinement phase. Recently, Mayer et al. [60] published HYPE, a single-level algorithm for connectivity optimization that grows \( k \) blocks using a hypergraph-specific neighborhood expansion [81] heuristic.

**Flows on Hypergraphs.** While flow-based approaches have not yet been considered as refinement algorithms for multilevel HGP, several works deal with flow-based hypergraph min-cut computation. The problem of finding minimum \((s, t)\)-cuts in hypergraphs was first considered by Lawler [54], who showed that it can be reduced to computing maximum flows in directed graphs. Hu and Moerder [48] present an augmenting path algorithm to compute a minimum-weight node separator on the star-expansion of the hypergraph. Their node-capacitated network can also be transformed into an edge-capacitated network using a transformation due to Lawler [55]. Yang and Wong [80] use repeated, incremental max-flow min-cut computations on the Lawler network [54] to find \( \varepsilon \)-balanced hypergraph bipartitions. Solution quality and running time of this algorithm are improved by Li et al. [57] by introducing advanced heuristics to select source and sink nodes. Furthermore, they present a preflow-based [38] min-cut algorithm that implicitly operates on the star-expanded hypergraph. Pistorius and Minoux [64] generalize the algorithm of Edmonds and Karp [28] to hypergraphs by labeling both vertices and nets. Liu and Wong [58] simplify Lawler’s hypergraph flow network [54] by explicitly distinguishing between graph edges and hyperedges with three or more pins. This approach significantly reduces the size of flow networks derived from VLSI hypergraphs, since most of the nets in a circuit are graph edges.
KaHyPar. Since our algorithm is integrated into the KaHyPar framework, we briefly review its core components. While traditional multilevel HGP algorithms contract matchings or clusterings and therefore work with a coarsening hierarchy of $O(\log n)$ levels, KaHyPar instantiates the multilevel paradigm in the extreme $n$-level version, removing only a single vertex between two levels. After coarsening, a portfolio of simple algorithms is used to create an initial partition of the coarsest hypergraph. During uncoarsening, strong localized local search heuristics based on the FM algorithm [29, 66] are used to refine the solution. KaHyPar supports both recursive bisection-based [69] and direct $k$-way [2] partitioning and allows for optimizing the cut-net as well as the connectivity metric. Our work builds on the direct $k$-way partitioning algorithm KaHyPar-CA [46], which uses an improved coarsening scheme that incorporates global information about the community structure of the hypergraph into the coarsening process. KaHyPar-CA has furthermore recently been embedded into an evolutionary framework: KaHyPar-E [6] is the first memetic HGP algorithm that uses the multilevel paradigm to effectively exploit the local solution space.

3 IMPROVED FLOW-BASED REFINEMENT FOR GRAPH PARTITIONING

We discuss the KaFFPa framework [67] in greater detail, since our work makes use of the techniques proposed by the authors. We furthermore identify shortcomings of the KaFFPa approach that restrict the search space of feasible solutions and propose a modification to overcome these limitations.

3.1 The KaFFPa Framework

Given an $\epsilon$-balanced $k$-way partition $\Pi_k = \{V_1, \ldots, V_k\}$ of a graph $G = (V, E, c, \omega)$, KaFFPa’s flow-based refinement algorithm works on pairs $(V_i, V_j)$ of adjacent blocks. To coordinate these refinements, the authors propose an active block scheduling algorithm, which schedules blocks adjacent in the quotient graph $Q$ as long as their participation in a pairwise refinement step improves solution quality or imbalance.

The basic idea is to build a flow network $N$ based on the induced subgraph $G[B]$, where $B \subseteq \{V_i, V_j\}$ is a set of nodes around the cut of the bipartition $\Pi_2 := \{V_i, V_j\}$. The size of $B$ is controlled by an imbalance factor $\epsilon' := \alpha \epsilon$, where $\alpha$ is a scaling parameter that is chosen adaptively depending on the result of the min-cut computation. If the heuristic found an $\epsilon$-balanced partition using $\epsilon'$, the cut is accepted and $\alpha$ is increased to $\min(2\alpha, \alpha')$ where $\alpha'$ is a predefined upper bound. Otherwise, it is decreased to $\max(\frac{\alpha'}{2}, 1)$. This scheme continues until a maximal number of rounds is reached or a feasible partition that did not improve the cut is found.

In each round, the corridor $B := B_i \cup B_j$ is constructed by performing two restricted breadth-first searches (BFS). The first BFS is done in the induced subgraph $G[V_i]$. It is initialized with the boundary nodes of $V_j$ and stops if $c(B_i)$ would exceed $(1 + \epsilon')\left[\frac{c(V_j)}{k}\right] - c(V_j)$. The second BFS constructs $B_j$ in an analogous fashion using $G[V_j]$. Let $B^- := \{u \in B \mid \exists (u, v) \in E : v \notin B\}$ be the border of $B$. Then $N$ is constructed by connecting all border nodes $B^- \cap V_i$ of $G[B]$ to an additional source node $s$ and all border nodes $B^- \cap V_j$ to an additional sink node $t$ using directed edges with an edge weight of $\infty$. By connecting $s$ and $t$ to the respective border nodes, it is ensured that edges incident to border nodes, but not contained in $G[B]$, cannot become cut edges. All edges of $G[B]$ use the corresponding edge weights of $G$ as capacities. An example is shown in Figure 1. For $\alpha = 1$, the size of $B$ ensures that $N$ has the cut property, i.e., each $(s, t)$-min-cut in $N$ yields a possibly smaller cut in $G$ that is feasible with respect to the original balance constraint of the $k$-way partition. For larger values of $\alpha$, this does not have to be the case.

Most Balanced Minimum Cuts. After computing a max-flow in $N$, the algorithm tries to find a min-cut with better balance. This is done by exploiting the fact that one $(s, t)$-max-flow contains
because they are locked in block via infinite-capacity edges. Our improved flow network is shown on the right. All nodes of all:

\[ S \subseteq \{ \epsilon \} \text{ or } S \in N \cap E \text{ or } u \in \text{a min-cut along either side } \Gamma(D) \text{ of } \]

is connected to border nodes. Furthermore, instead of using infinite capacity edges to all nodes:

\[ V \cap \{ \text{cut-border edges } S \} \text{ are computed by } \]

is constructed by contracting each strongly connected component of the residual graph. Then, the following heuristic (called most balanced minimum cuts) is repeated several times using different random seeds. Closed sets containing \( s \) are computed by sweeping through the nodes of \( D_{s,t} \) in reverse topological order (e.g., computed using a randomized depth-first search (DFS)). Each closed set induces a differently balanced min-cut and the one with the best balance (with respect to the original balance constraint) is used to improve the cut and/or balance between \( V_i \) and \( V_j \).

### 3.2 Improving the Flow Network

In the following, we distinguish between peripheral edges \( E^- := \{(u, v) \in E : u \in B^- \land v \notin \Pi_2 \} \) and border edges \( E^+ := \{(u, v) \in E : u \in B^- \land v \in \Pi_2 \setminus B \} \). By connecting the source \( s \) to all nodes \( S = B^- \cap V_i \) and all nodes \( T = B^- \cap V_j \) to the sink \( t \) using directed edges with infinite capacity, the KaFFPa network ensures that the max-flow computation does not affect border nodes and thus peripheral and border edges. However, this unnecessarily restricts the search space of feasible solutions. Since border nodes \( B^- \) are directly connected to either \( s \) or \( t \) via infinite capacity edges, every min-cut \( (S, B \setminus S) \) will have \( S \subseteq S \) and \( T \subseteq B \setminus S \). The KaFFPa model therefore prevents (i) all min-cuts in which any non-cut border edge becomes part of the cut-set, (ii) all cut-border edges from being removed from the cut-set, and (iii) all border nodes incident to peripheral but not to border edges from changing their block (see Figure 1 for an example). This restricts the space of possible solutions, since the corridor \( B \) was computed such that even a min-cut along either side of the border would result in a feasible cut in \( G \). Thus, ideally, all vertices \( v \in B \) should be able to change their block as a result of an \( (s, t) \)-max-flow computation on \( N \)—not only vertices \( v \in B \setminus B^- \). This limitation becomes increasingly relevant for graphs with high average node degrees as well as for partitioning problems with small imbalance \( \varepsilon \), since high-degree nodes \( u \) are likely to have some neighbors \( \Gamma(u) \notin G[B] \) (i.e., some of their incident edges are either border or peripheral edges) and tight balance constraints enforce small \( B \)-corridors.

We overcome these restrictions by treating border nodes differently. Since peripheral edges are cut edges in the \( k \)-way partition of \( G \) that are not affected if incident border nodes \( u \in G[B] \) change their block (i.e., they will remain cut edges in \( G \)), border nodes only incident to peripheral edges are neither connected to \( s \) nor \( t \). Furthermore, instead of using infinite capacity edges to connect the source or the sink to the remaining border nodes, we use the actual edge weights of incident border edges as capacities. More precisely, \( s \) is connected to border nodes \( u \in B^- \) using information about all \((s, t)\)-min-cuts [63]. More precisely, the algorithm uses the 1–1 correspondence between \((s, t)\)-min-cuts and closed sets containing \( s \) in the Picard-Queyranne-DAG \( D_{s,t} \) of the residual graph \( K \) [63]. First, \( D_{s,t} \) is constructed by contracting each strongly connected component of the residual graph. Then, the following heuristic (called most balanced minimum cuts) is repeated several times using different random seeds. Closed sets containing \( s \) are computed by sweeping through the nodes of \( D_{s,t} \) in reverse topological order (e.g., computed using a randomized depth-first search (DFS)). Each closed set induces a differently balanced min-cut and the one with the best balance (with respect to the original balance constraint) is used to improve the cut and/or balance between \( V_i \) and \( V_j \).
an edge \((s, u)\) with capacity \(c(s, u) := \sum_{(\Gamma(u) \setminus B) \cap V_i} \omega(u, v)\) and \(u\) is connected to \(t\) using an edge \((u, t)\) with capacity \(c(u, t) := \sum_{(\Gamma(u) \setminus B) \cap V_j} \omega(u, v)\) (see Figure 1 for an example). Since it is now possible for a max-flow computation to saturate border edges, we get a flow network that (i) does not lock any node \(u \in G[B]\) in its block and (ii) correctly models the impact of the max-flow min-cut computation on the solution quality of the \(k\)-way partition of \(G\).

4 HYPERGRAPH MAX-FLOW MIN-CUT REFINEMENT

In the following, we generalize the KaFFPa algorithm to hypergraph partitioning. In Section 4.1, we first show how hypergraph flow networks \(N\) are constructed in general and introduce a technique to reduce their size by removing low-degree vertices. Given a \(k\)-way partition \(\Pi_k = \{V_1, \ldots, V_k\}\) of a hypergraph \(H\), a pair of blocks \((V_i, V_j)\) adjacent in the quotient graph \(Q\), and a corridor \(B\), Section 4.2 then explains how \(N\) is used to build a flow problem \(F\) based on the subhypergraph \(H_B = (V_B, E_B)\). By connecting the source node \(s\) and the sink node \(t\) to specific nodes of the flow network, \(F\) is constructed such that an \((s, t)\)-max-flow computation optimizing the cut-net metric in the bipartition \(\Pi_2 = (V_i, V_j)\) of \(H_B\) improves either the cut-net or the connectivity metric in the \(k\)-way partition of \(H\). Section 4.3 then discusses the integration into KaHyPar and introduces several techniques to speed up flow-based refinement. A pseudocode description of the entire flow-based refinement framework is given in Algorithm 1.

\begin{algorithm}
\caption{Flow-Based Refinement}
\begin{algorithmic}[1]
\Require Hypergraph \(H\), \(k\)-way partition \(\Pi_k = \{V_1, \ldots, V_k\}\), imbalance parameter \(\epsilon\).
\Ensure improved \(\epsilon\)-balanced \(k\)-way partition \(\Pi_k = \{V_1, \ldots, V_k\}\).
\Comment in the beginning all blocks are active
\Procedure{Algorithm}{MaxFlowMinCutRefinement}(\(H, \Pi_k\))
\Comment choose a pair of blocks
\While {\exists active blocks \(\in Q\)}
\State \(\Pi_{old} := \Pi_{best} := \{V_i, V_j\} \subseteq \Pi_k\)
\Comment imbalance of current \(k\)-way partition
\State \(\epsilon_{old} := \epsilon_{best} := \text{imbalance}(\Pi_k)\)
\Comment use large \(B\)-corridor for first iteration
\Comment adaptive flow iterations
\Do
\State \(B := \text{computeB-Corridor}(H, \Pi_{best}, \alpha \epsilon)\)
\Comment as described in Section 3
\State \(H_B := \text{SubHypergraph}(H, B)\)
\Comment create subhypergraph
\State \(N_B := \text{FlowNetwork}(H_B)\)
\Comment as described in Section 4.1
\State \(F := \text{FlowProblem}(N_B)\)
\Comment as described in Section 4.2
\State \(f := \text{maxFlow}(F)\)
\Comment compute maximum flow on \(F\)
\State \(\Pi_f := \text{mostBalancedMinCut}(f, F)\)
\Comment as in Section 3 & 4.1
\Comment imbalance of new \(k\)-way partition
\State \(\epsilon_f := \text{imbalance}(\Pi_f \cup \Pi_k \setminus \Pi_{old})\)
\If {\((\text{cut}(\Pi_f) < \text{cut}(\Pi_{best}) \land \epsilon_f \leq \epsilon) \lor \epsilon_f < \epsilon_{best}\)}
\Comment found improvement
\State \(\alpha := \min(2\alpha, \alpha')\), \(\Pi_{best} := \Pi_f, \epsilon_{best} := \epsilon_f\)
\Comment update \(\alpha, \Pi_{best}, \epsilon_{best}\)
\Else
\State \(\alpha := \frac{\alpha}{2}\)
\Comment decrease size of \(B\)-corridor in next iteration
\EndIf
\EndDo
\If {\(\Pi_{best} \neq \Pi_{old}\)}
\Comment improvement found
\State \(\Pi_k := \Pi_{best} \cup \Pi_k \setminus \Pi_{old}\)
\Comment replace \(\Pi_{old}\) with \(\Pi_{best}\)
\State \(\text{activateForNextRound}(V_i, V_j)\)
\Comment reactivate blocks for next round
\EndIf
\EndWhile
\Return \(\Pi_k\)
\EndProcedure
\end{algorithmic}
\end{algorithm}
4.1 Hypergraph Flow Networks

From Hypergraphs to Flow Networks. Given a hypergraph $H = (V, E, c, \omega)$ and two distinct vertices $s$ and $t$, we first reduce the problem of finding an $(s, t)$-min-cut in $H$ to the problem of finding a minimum-weight $(s, t)$-node-separator in the star-expansion $G_*$, where each star-node $e$ has weight $c(e) = \omega(e)$ and all other nodes $v$ have a weight of infinity [48]. This network is then transformed into the edge-capacitated flow network $N = (V, E, c)$ of Lawler [54] as follows: $V$ contains all non-star nodes $v$. For each star-node $e$, add two bridging nodes $e'$ and $e''$ to $V$ and a bridging edge $(e', e'')$ with capacity $c(e', e'') = c(e)$ to $E$. For each neighbor $u \in \Gamma(e)$, add two edges $(u, e')$ and $(e'', u)$ with infinite capacity to $E$. The size of this network can be reduced by distinguishing between star-nodes corresponding to multi-pin nets and those corresponding to two-pin nets in $H$. In the flow network of Liu and Wong [58], the former are transformed as described above, while the latter (i.e., star-nodes $e$ with $|\Gamma(e)| = |\{u, v\}| = 2$) are replaced with two edges $(u, v)$ and $(v, u)$ with capacity $c(e)$. For each such star-node, this decreases the network size by two nodes and three edges. Figure 2 shows both networks as well as ours, which we describe in the following.

Removing Low-Degree Vertices. We further decrease the network size by using the observation that the $(s, t)$-node-separator in $G_*$ has to be a subset of the star-nodes, since all other nodes have infinite capacity. Thus, it is possible to replace any infinite-capacity node by adding a clique between all adjacent star-nodes without affecting the separator. The key observation now is that an infinite-capacity node $v$ with degree $d(v)$ induces $2d(v)$ edges in the Lawler network [54], while a clique between star-nodes induces $d(v)(d(v) - 1)$ edges. For non-star nodes $v$ with $d(v) \leq 3$, it therefore holds that $d(v)(d(v) - 1) \leq 2d(v)$. We therefore remove all infinite-capacity nodes $v$ corresponding to hypernodes with $d(v) \leq 3$ that are not incident to any two-pin nets by adding a clique between all star-nodes $\Gamma(v)$. In case $v$ was either source or sink, we create a multi-source multi-sink problem by adding the star-nodes $\Gamma(v)$ to the set of sources resp. sinks [31]. We then apply the transformation of Liu and Wong [58].

Reconstructing Min-Cuts. After computing an $(s, t)$-max-flow $f$ in the Lawler or Liu-Wong network, an $(s, t)$-min-cut of $H$ can then be computed by a BFS in the residual graph $N_f$ starting from $s$ [54]. Let $S$ be the set of nodes corresponding to vertices of $H$ reached by the BFS. Then, $(S, V \setminus S)$ is an $(s, t)$-min-cut. Since our network does not contain low-degree vertices, we use the following lemma to compute an $(s, t)$-min-cut of $H$:

**Lemma 4.1.** Let $f$ be a maximum $(s, t)$-flow in the Lawler network $N = (V, E)$ of a hypergraph $H = (V, E)$ and $(S, V \setminus S)$ be the corresponding $(s, t)$-min-cut in $N$. Then, for each node $v \in S \cap V$, the residual graph $N_f = (V_f, E_f)$ contains at least one path $(s, \ldots, e'')$ to a bridging node $e''$ of a net $e \in \Gamma(v)$.
PROOF. Since \( v \in S \), there has to be some path \( s \sim v \) in \( N_f \). By definition of the flow network, this path can either be of the form \( P_1 = (s, \ldots, e'', v) \) or \( P_2 = (s, \ldots, e', v) \) for some bridging nodes \( e', e'' \) corresponding to nets \( e \in I(v) \). In the former case, we are done, since \( e'' \in P_1 \). In the latter case, the existence of edge \((e', v) \in E_f\) implies that there is a positive flow \( f(e', v) > 0 \) over edge \((v, e') \in E\). Due to flow conservation, there exists at least one edge \((e'', v) \in E\) with \( f(e'', v) > 0 \), which implies that \((v, e'') \in E_f\). Thus we can extend the path \( P_2 \) to \((s, \ldots, e', v, e'')\).

Thus, \((A, V \setminus A)\) is an \((s, t)\)-min-cut of \( H \), where \( A := \{v \mid \exists e \in E : v \in e \land (s, \ldots, e'') \in N_f\} \). Furthermore this allows us to employ the most balanced minimum cut heuristic as described in Section 3.1. By the definition of closed sets, it follows that if a bridging node \( e'' \) is contained in a closed set \( C \), then all nodes \( v \in \Gamma(e'') \) (corresponding to vertices of \( H \)) are also contained in \( C \). Thus, we can use the respective bridging nodes \( e'' \) as representatives of removed low-degree vertices.

4.2 Constructing the Hypergraph Flow Problem

Let \( H_B = (V_B, E_B) \) be the subhypergraph of \( H = (V, E) \) that is induced by a corridor \( B \) computed in the bipartition \( \Pi_2 = (V_i, V_j) \). In the following, we distinguish between the set of \textit{inner} border nodes \( B^\rightarrow := \{v \in V_B \mid \exists e \in E : \{u, v\} \subseteq e \land u \notin V_B\} \), the set of \textit{outer} border nodes \( B^\leftarrow := \{u \in \Pi_2 \setminus V_B \mid \exists e \in E : \{u, v\} \subseteq e \land v \in V_B\} \), and the set of \textit{peripheral} nodes \( B^\circ := \{u \notin \Pi_2 \mid \exists e \in E : \{u, v\} \subseteq e \land v \in V_B\} \). Similarly, we now distinguish between \textit{outer} nets \( E^\leftarrow := \{e \in E : e \cap V_B = \emptyset\} \) with no pins inside \( H_B \), \textit{inner} nets \( E^\rightarrow := \{e \in E : e \cap V_B = e\} \) with all pins inside \( H_B \), the set of \textit{border} nets \( E_B^\circ := \{e \in E : e \in I(B^\rightarrow) \cap I(B^\leftarrow)\} \), and the set of \textit{peripheral} nets \( E^\circ := \{e \in E : e \in I(B^\rightarrow) \cap I(V \setminus \Pi_2)\} \). A visualization of these definitions is shown in Figure 3.

A hypergraph flow problem \( F \) consists of a flow network \( N_B = (V_B, E_B) \) derived from \( H_B \) and two \textit{additional} nodes \( s \) and \( t \) that are connected to some nodes \( v \in V_B \). It has the cut property if the max-flow induced min-cut bipartition \( \Pi_f \) of \( H_B \) does not worsen the partitioning objective in \( H \). For both the cut-net and the connectivity objective, it thus has to hold that \( \text{cut}(\Pi_f) \leq \text{cut}(\Pi_2) \), since \( \lambda^{-1}(\Pi) = \text{cut}(\Pi) \) for bipartitions. While outer nets are not affected by a max-flow computation, the max-flow min-cut theorem [32] ensures the cut property for all inner nets. Peripheral and border nets, however, require special attention. Since these nets are only \textit{partially} contained in \( H_B \) and \( N_B \), they will remain connected to all blocks \( \Lambda(e) \setminus \{V_i, V_j\} \) regardless of the result of the max-flow computation. It is, therefore, necessary to "encode" information about peripheral and border nets into the flow problem. For simplicity, we first discuss how this is done using the traditional KaFFPa approach described in Section 3.1 and then show how the improved flow model presented in Section 3.2 can be generalized from graphs to hypergraphs.

Cut-Net Optimization. In graph partitioning, peripheral edges are not part of the induced subgraph \( G[B] \) and, thus, not contained in the resulting flow network since they have no influence on the edge cut of the bipartition \( \Pi_2 \). In hypergraph partitioning, however, peripheral nets \( e \in E^\circ \)
are partially contained in the subhypergraph $H_B$. Since $\Lambda(e) \setminus \Pi_2 \neq \emptyset$ for these nets, a max-flow min-cut computation in the flow network of $H_B$ cannot remove them from the cut-set of the $k$-way partition $\Pi_k$. To account for that fact, we remove all peripheral nets from $H_B$ before constructing the hypergraph flow network $\mathcal{N}_B$. Border nets $e \in E^\perp$ on the other hand remain connected to the blocks of their outer border nodes in $\Pi_2$. A special case unique to hypergraphs are border nets $e$ that are connected to both $V_i$ and $V_j$ by some outer border nodes $B^\perp \cap e$. As with peripheral nets, a max-flow computation in the flow network of $H_B$ will not be able to remove these nets (i.e., border nets $e \in E^\perp : \Phi(e, V_i \setminus B_i) \geq 1 \land \Phi(e, V_j \setminus B_j) \geq 1$) from the cut, since they are locked in the cut-set of $\Pi_2$. We therefore remove them from $H_B$ along with the peripheral nets before constructing the flow network $\mathcal{N}_B$. To account for the remaining border nets that are only connected to either $V_i$ or $V_j$, we generalize the KaFFPa approach by connecting $s$ to all nodes $S = B^\perp \cap V_i$ if $\Phi(e, V_i \setminus B_i) \geq 1$ and all nodes $T = B^\perp \cap V_j$ to $t$ if $\Phi(e, V_j \setminus B_j) \geq 1$ using directed edges with infinite capacity.

**Connectivity Optimization.** While border nets are treated in the same way as for cut-net optimization, we do not remove peripheral nets $e \in E^\perp$ from $H_B$ before constructing the flow network $\mathcal{N}_B$ when optimizing the connectivity metric. Since these nets are partially contained in $H_B$, a max-flow min-cut computation in $\mathcal{N}_B$ can remove them from the cut-set of $\Pi_k$. This decreases the connectivity $\lambda(e)$ and thus improves the overall solution quality $\lambda^{-1}(\Pi_k)$ of the $k$-way partition.

**Improving the Model.** We exploit the structure of hypergraph flow networks such that $(s, t)$-max-flow computations can also cut through non-cut border nets. The limitation of the original KaFFPa model becomes increasingly relevant for hypergraph partitioning, as large nets are likely to be only partially contained in $H_B$ and thus likely to be border nets. Instead of directly connecting $s$ and $t$ to inner border nodes $B^\perp$ as described above and thus preventing all min-cuts in which these nodes switch blocks, we conceptually extend $H_B$ to contain all outer border nodes $B^\perp$ and the remaining border nets $E^\perp_B$. The resulting hypergraph is $H^\perp_B = (V_B \cup B^\perp, \{e \in E \mid e \cap V_B \neq \emptyset\})$.

The key insight now is that by using the flow network of $H^\perp_B$ and connecting $s$ resp. $t$ to the outer border nodes $B^\perp \cap V_i$ resp. $B^\perp \cap V_j$, we get a flow problem that does not lock any node $v \in V_B$ in its block, since none of them is directly connected to either $s$ or $t$. Due to the max-flow min-cut theorem [32], this flow problem has the cut property, since all border nets of $H_B$ are now internal nets and all external border nodes $B^\perp$ are locked inside their block. However, it is not necessary to use $H^\perp_B$ instead of $H_B$ to achieve this result. For all $v \in B^\perp$, the flow network of $H^\perp_B$ contains paths $(s, v, e')$ and $(e'', v, t)$ that only involve infinite-capacity edges. Therefore, we can remove all nodes $v \in B^\perp$ by directly connecting $s$ and $t$ to the corresponding bridging nodes $e', e''$ via infinite-capacity edges without affecting the maximal flow [31]. More precisely, in the flow problem $\mathcal{F}_H$, we connect $s$ to all bridging nodes $e'$ corresponding to border nets $e \in E^\perp_B : e \subset B^\perp \cap V_i$ and all bridging nodes $e''$ corresponding to border nets $e \in E^\perp_B : e \subset B^\perp \cap V_j$ to $t$ using directed, infinite-capacity edges.

**Single-Pin Border Nets.** Border nets with $|e \cap B^\perp| = 1$ can furthermore be modeled more efficiently. For such a net $e$, the flow problem contains paths of the form $(s, e', e'', v)$ or $(v, e', e'', t)$, which can be replaced by paths of the form $(s, e', v)$ or $(v, e'', t)$ with $c(e', v) = \omega(e)$ resp. $c(v, e'') = \omega(e)$. In both cases, we can thus remove one bridging node and two infinite-capacity edges. A comparison of the original KaFFPa flow problem $\mathcal{F}_G$ and our improved version $\mathcal{F}_H$ is shown in Figure 4.

### 4.3 Implementation Details

Since KaHyPar is an $n$-level partitioner, its FM-based local search algorithms are executed each time a vertex is uncontracted. To prevent expensive recalcations, it therefore uses a cache to maintain the gain values of FM moves throughout the $n$-level hierarchy [2]. In order to combine...
Fig. 4. Comparison of the KaFFPa [67] flow problem $F_G$ and our flow problem $F_H$ for bipartitioning. For clarity, the zoomed-in view is based on the Lawler network.

our algorithm with FM local search, we not only perform the moves induced by the max-flow min-cut computation but also update the FM gain cache accordingly. Since it is not feasible to execute our algorithm on every level of the $n$-level hierarchy, we use an exponentially spaced approach that performs flow-based refinements after uncontracting $i = 2^j$ vertices for $j \in \mathbb{N}_+$. This way, the algorithm is executed more often on smaller flow problems than on larger ones. To further improve the running time, we introduce the following speedup techniques: We modify active block scheduling such that after the first round the algorithm is only executed on a pair of blocks if at least one execution using these block led to an improvement on previous levels (S1). We skip flow-based refinement if the cut between two adjacent blocks is less than 10 on all levels except the finest (S2). We stop resizing the $B$-corridor if the current cut did not improve the previously best solution (S3).

5 EXPERIMENTAL EVALUATION

We implemented the max-flow min-cut refinement algorithm in the hypergraph partitioning framework KaHyPar. Our implementation is available from \url{http://www.kahypar.org}. In the experimental evaluation, we equip different KaHyPar configurations with our flow-based refinement framework. The suffix -MF is used to distinguish configurations using (M)aximum (F)lows from those where flow-based refinements are disabled. Furthermore, we implemented the improvements for graph partitioning described in Section 3.2 in KaFFPa\(^1\) [68] (KaHIP v2.0). In both cases, the code is written in C++ and compiled using g++-5.2 with flags -O3 -march=native.

Outline. After discussing our experimental setup in Section 5.1, we evaluate the different hypergraph flow networks, our improved flow model, and two state-of-the-art max-flow algorithms within our framework in Section 5.2. In Section 5.3, we then discuss different framework configurations, before comparing the final configuration with the previously best versions of KaHyPar for cut-net and connectivity optimization in Section 5.4. Afterward, Sections 5.5 and 5.6 compare the flow-based KaHyPar configurations to state-of-the-art hypergraph partitioning systems for connectivity and cut-net optimization. In Section 5.7, we briefly turn to edge-cut optimization for graph partitioning in order to (i) validate that our improvements to KaFFPa’s flow network proposed in Section 3.2 translate into improved solution quality and (ii) to compare the performance of KaHyPar-MF with KaFFPa—the only multilevel graph partitioning algorithm that also combines FM-based local search with flow-based refinements. Lastly, we integrate our refinement framework into the memetic multilevel hypergraph partitioning algorithm KaHyPar-E [6] and evaluate the performance of KaHyPar-E-MF in Section 5.8.

---

\(^1\)Available from \url{https://github.com/SebastianSchlag/KaHIP/tree/kahypar_flow_model}.
Table 1. Overview about Different Benchmark Sets

| Source | DAC2012 | ISPD98 | Primal | Dual | Literal | SPM | Graphs |
|--------|---------|--------|--------|------|---------|-----|-------|
| Set A  | [46]    | 488    | 10     | 18   | 92      | 92  | 184   |-      |
| Set B  | [46]    | 165    | 5      | 10   | 30      | 30  | 60    |-      |
| Set C  | [6]     | 100    | 4      | 10   | 18      | 18  | 32    |-      |
| Set D  | [6, 45] | 25     | -      | 5    | 5       | 5   | 5     | 21    |

Sets B, C, and D are subsets of set A.

5.1 Experimental Setup

**Instances.** Table 1 gives an overview about the benchmark sets used in the experiments. The benchmark set of Heuer and Schlag\(^2\) [46] is used as the main benchmark set and referred to as set A. It contains 488 hypergraphs derived from four benchmark sets: the ISPD98 VLSI Circuit Benchmark Suite [3], the DAC 2012 Routability-Driven Placement Contest [78], the SuiteSparse Matrix Collection [22], and the international SAT Competition 2014 [9]. 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. SAT instances are converted to three different representations: For literal hypergraphs, each Boolean literal is mapped to one vertex and each clause constitutes a net [62], while in the primal model each variable is represented by a vertex and each clause forms a net. In the dual model, the opposite is the case [59]. All hypergraphs have unit vertex and net weights.

We furthermore use the representative subset of set A proposed in Ref. [46] (165 hypergraphs, denoted as set B) and the subset of 100 hypergraphs used in the evaluation of KaHyPar-E [6] (set C). Based on an experiment to estimate the number of hypergraphs necessary to produce the same qualitative results as the entire benchmark set, set B was initially chosen in Ref. [69] to contain the 10 largest ISPD98 VLSI hypergraphs, 30 randomly chosen SAT hypergraphs (literal representation), and 60 randomly chosen sparse matrix hypergraphs. In order to incorporate more recent VLSI circuits and more common SAT models, it was then extended in Ref. [46] to also contain the five smallest DAC2012 circuits, as well as the primal and dual representations of each literal SAT hypergraph. Set C was composed in Ref. [6] such that it includes hypergraphs from all instance classes and such that all hypergraphs could be partitioned within the given time limit.

To evaluate the effects of our improved flow-model on both hypergraph and graph partitioning, we furthermore use a smaller subset consisting of 25 hypergraphs (set D) and the 21 large web graphs and social networks from Ref. [61] (set E). Set D was introduced in the conference version of this work [45] and was also used as parameter tuning benchmark set for KaHyPar-E [6]. It was chosen to contain five small to medium-sized hypergraphs from each instance classes except DAC2012, for which all instances were considered too large. Basic properties of sets D and E can be found in Tables 10 and 11 in Appendix A.

Unless mentioned otherwise, all hypergraphs and graphs are partitioned with an allowed imbalance of \(\varepsilon = 0.03\) into \(k \in \{2, 4, 8, 16, 32, 64, 128\}\) blocks. For each value of \(k\), a \(k\)-way partition is considered to be one test instance, resulting in a total of 3,416 instances for set A, 1,155 instances for set B, 700 instances for set C, 175 for set D, and 147 instances for set E.

**System and Methodology.** We compare different KaHyPar configurations to the \(k\)-way (hMetis-K) and recursive bisection variants (hMetis-R) of hMetis 2.0 (p1) [51, 52], PaToH 3.2 [16], Zoltan-AlgD [71], and HYPE [60]. We choose these tools because of the following reasons: PaToH produces better quality than Zoltan’s native parallel hypergraph partitioner (PHG) in serial mode [11, 25].

\(^2\)The benchmark set and detailed statistics for each hypergraph are available from [http://algo2.iti.kit.edu/schlag/sea2017/](http://algo2.iti.kit.edu/schlag/sea2017/).
Parkway does not run in serial mode and was found to be comparable to Zoltan’s PHG in serial mode [25]. The algebraic multigrid coarsening algorithm of Zoltan-AlgD has been shown to improve the performance of Zoltan’s PHG in serial mode for the cut-net metric [71]. Furthermore, PaToH has been shown to compute better solutions than Mondriaan [10, 65] and MLPart [17]. Additionally, MLPart is restricted to bisections [14, 18]. The performance of SHP is comparable to the performance of Zoltan and Mondriaan [50]. UMPa does not improve on PaToH when optimizing single objective functions that do not benefit from the directed hypergraph model [24]. Furthermore, kPaToH [7] did not perform better than PaToH in preliminary experiments [2]. Lastly, HYPE so far has only been evaluated on four hypergraphs and only been compared to hMetis-R [60].

Note that all multilevel hypergraph partitioning systems have a considerable number of tuning parameters and configurable subroutines, many of which interact in nontrivial ways. We therefore refrain from tuning these systems ourselves and instead use the configurations provided by the authors, which have been shown to work well in the corresponding publications.

For connectivity optimization, we use the results of KaHyPar-CA, hMetis-R, hMetis-K, and PaToH using the default preset (PaToH-D) reported by Heuer and Schlag [46]. All comparisons involving set A, therefore, are based on 3,222 instances, since 194 out of 3,416 instances were already excluded in Ref. [46] because either PaToH-Q could not allocate enough memory or other partitioners did not finish in time. The experimental data contains the arithmetic mean of the computed connectivities and running times as well as the best connectivity found for 10 repetitions with different seeds per instance. Each partitioner had a time limit of 8 hours per instance and seed. Since PaToH previously ignored the seed if configured to use the quality preset (PaToH-Q) [46], we do not use the PaToH-Q results presented in Ref. [46]. Instead, we redo all PaToH-Q experiments from scratch and now report the results of ten repetitions using different seeds. For HYPE, we report the results of one iteration using the default configuration (which is not randomized and was also used for the experiments in Ref. [60]), since employing randomization did not improve solution quality (see Appendix B).

For cut-net optimization, we conduct new experiments on set A for all algorithms involved in the evaluation. We use the same system that is used in the experiments of Heuer and Schlag [46] for all new experiments reported in this article. Each partitioner runs on a single node of a cluster with 512 nodes. Each node has two Intel Xeon E5-2670 Octa-Core (Sandy Bridge) processors clocked at 2.6GHz, 64GB main memory, 20MB L3- and 8×256KB L2-Cache and runs Red Hat Enterprise Linux 7.4. Unless mentioned otherwise, we use the same number of repetitions and the same time limit of 8 hours.

When computing averages over different instances (solution quality, running time), we use the geometric mean in order to give every instance a comparable influence on the final result.

Performance Profiles. In order to compare different algorithms in terms of solution quality, we perform a more detailed analysis using performance profiles [26]. For a set of \( \mathcal{P} \) algorithms and a benchmark set \( \mathcal{I} \) containing \( |\mathcal{I}| \) instances, the performance ratio \( r_{p,i} \) relates the cut computed by partitioner \( p \) for instance \( i \) to the smallest minimum cut of all algorithms, i.e.,

\[
r_{p,i} := \frac{\text{cut}_{p,i}}{\min\{\text{cut}_{p,i} : p \in \mathcal{P}\}}.
\]

The performance profile \( \rho_p(\tau) \) of algorithm \( p \) is then given by the function

\[
\rho_p(\tau) := \frac{|\{i \in \mathcal{I} : r_{p,i} \leq \tau\}|}{|\mathcal{I}|}, \tau \geq 1.
\]

Available from: http://algo2.iti.kit.edu/schlag/sea2017/.

Detailed per-instance results of all experiments can be found online: https://algo2.iti.kit.edu/schlag/jea2019/.
Table 2. Statistics of Benchmark Set B

| Type | #   | $d(v)$ | $d(v)$ | $|e|$ | $|e|$ |
|------|-----|--------|--------|------|------|
| DAC  | 5   | 3.32   | 3.28   | 3.37 | 3.35 |
| ISPD | 10  | 4.20   | 4.24   | 3.89 | 3.90 |
| Primal | 30   | 16.29 | 9.97   | 2.63 | 2.39 |
| Literal | 30   | 8.21  | 4.99   | 2.63 | 2.39 |
| Dual | 30  | 2.63   | 2.38   | 16.29 | 9.97 |
| SPM  | 60  | 24.78  | 14.15  | 26.58 | 15.01 |

We use $\overline{x}$ to denote mean and $\tilde{x}$ to denote the median.

For connectivity optimization, the performance ratios are computed using the connectivity values $\lambda^{-1}$ instead of the cut values. The value of $\rho_p(1)$ corresponds to the fraction of instances for which partitioner $p$ computed the best solution, while $\rho_p(\tau)$ is the probability that a performance ratio $r_{p,i}$ is within a factor of $\tau$ of the best possible ratio. Note that since performance profiles only allow to assess the performance of each algorithm relative to the best algorithm, the $\rho(1)$ values cannot be used to rank algorithms (i.e., to determine which algorithm is the second best, and so on) [39]. In our experimental analysis, the performance profile plots are based on the best solutions (i.e., minimum connectivity/cut) each algorithm found for each instance. Furthermore, we choose parameters $r_{\text{inf}} \gg r_{p,i}$ for all $p, i$ and $r_{\text{timeout}} \gg r_{\text{inf}}$ such that a performance ratio $r_{p,i} = r_{\text{inf}}$ if and only if algorithm $p$ computed an infeasible solution for instance $i$, and $r_{p,i} = r_{\text{timeout}}$ if and only if the algorithm could not compute a solution for instance $i$ within the given time limit. Since the performance ratios are heavily right-skewed, the performance profile plots are divided into three segments with different ranges for parameter $\tau$ to reflect various areas of interest. The first segment highlights small values ($\tau \leq 1.1$), while the second segment contains results for all instances that are up to a factor of $\tau = 2$ worse than the best possible ratio. The last segment contains all remaining ratios, i.e., instances for which some algorithms performed considerably worse than the best algorithm, instances for which algorithms produced infeasible solutions, and instances that could not be partitioned within the given time limit. The last segment uses a log-scale on the $x$-axis.

Statistical Significance. To determine whether or not the differences in solution quality of our proposed KaHyPar configurations and the other algorithms are statistically significant, we follow the guidelines proposed by Demsar [23] and García et al. [35, 36]. For comparisons involving two algorithms, we use the Wilcoxon signed rank test [47, 79]. To compare multiple algorithms with a control (i.e., a certain KaHyPar configuration), we use the Friedman test [33, 34] with Iman Davenport modification [49] and the Finner post-hoc procedure [15, 30]. In all cases, we use a 1% significance level.

5.2 Evaluating Flow Networks, Models, and Algorithms

Flow Networks and Algorithms. To analyze the effects of the different hypergraph flow networks, we compute five bipartitions for each hypergraph of set B with KaHyPar-CA using different seeds. Statistics of the hypergraphs are shown in Table 2. The bipartitions are then used to generate hypergraph flow networks for a corridor of size $|B| = 25,000$ hypernodes around the cut. Figure 5 (top) summarizes the sizes of the respective flow networks in terms of number of nodes $|V|$ and number of edges $|E|$ for each instance class. We consider the Lawler network $N_L$, the Liu-Wong network $N_W$, and our network $N_{\text{Our}}$. Furthermore, we show results for $N_{\text{Our}}^1$ that exploit the fact that the flow problems are based on subhypergraphs $H_B$ by additionally modeling single-pin border nets more efficiently as described in Section 4.2. We see that the flow networks of primal and literal SAT instances are the largest in terms of both numbers of nodes and edges. High average vertex degrees combined with low average net sizes lead to subhypergraphs $H_B$ containing many
Fig. 5. Top: Size of the flow networks when using the Lawler network $N_L$, the Liu-Wong network $N_W$, and our network $N_{\text{Our}}$. Network $N_{\text{Our}}^1$ additionally models single-pin border nets more efficiently. The dashed line indicates 25,000 nodes. Bottom: Speedup of BK $\text{[12]}$ and IBFS $\text{[37]}$ max-flow algorithms over the execution on the Lawler network $N_L$.

small nets, which then induce many nodes and (infinite-capacity) edges. Dual instances with low average degree and large average net size, on the other hand, lead to smaller flow networks. For VLSI instances (DAC, ISPD), both average degrees and average net sizes are low, while for Sparse Matrices (SPMs) hypergraphs, the opposite is the case. This explains why SPM flow networks have significantly more edges, despite the number of nodes being comparable in both classes.

As expected, the Lawler network $N_L$ induces the biggest flow problems. Looking at the Liu-Wong network $N_W$, we can see that distinguishing between graph edges and nets with $|e| \geq 3$ pins has an effect for all hypergraphs with many small nets (i.e., DAC, ISPD, Primal, Literal). While this technique alone does not improve dual SAT instances, we see that the combination of the Liu-Wong approach and the removal of low-degree hypernodes in $N_{\text{Our}}$ reduces the size of the networks for all instance classes except SPM. Both techniques only have a limited effect on these instances, since both hypernode degrees and net sizes are large on average. Using $N_{\text{Our}}^1$ further reduces the network sizes significantly. As expected, the reduction in numbers of nodes and edges is most pronounced for hypergraphs with low-average net sizes because these instances are likely to contain many single-pin border nets.

To further see how these reductions in network size translate to improved running times of max-flow algorithms, we use these networks to create flow problems using our flow model $F_H$ and compute min-cuts using two highly tuned max-flow algorithms, namely the BK-algorithm$^5$ [12] and the incremental breadth-first search (IBFS) algorithm$^6$ [37]. These algorithms were chosen

---

$^5$Available from: https://github.com/gerddie/maxflow.

$^6$Available from: http://www.cs.tau.ac.il/~sagihed/ibfs/code.html.
because they performed best in preliminary experiments [43]. We compare the speedups of these algorithms when executed on $N_W$, $N_{1\text{Our}}$, and $N_{1\text{Our}}$ to the execution on the Lawler network $N_L$ in Figure 5 (bottom). Both algorithms benefit from improved network models on all instance classes except SPM, and the speedups directly correlate with the reductions in network size. SPM hypergraphs have high average vertex degrees and large average net sizes, in which case the optimizations only have a very limited effect since they target small nets and low-degree vertices. While $N_W$ significantly reduces the running times for Primal and Literal instances, $N_{1\text{Our}}$ additionally leads to a speedup for Dual instances. By additionally considering single-pin border nets, $N_{1\text{Our}}$ results in an average speedup between 1.52 and 2.21 (except for SPM instances). Since IBFS outperformed the BK algorithm in Ref. [43], we use $N_{1\text{Our}}$ and IBFS in all following experiments.

Flow Models. We now compare KaFFPa’s flow model $\mathcal{F}_G$ to our model $\mathcal{F}_H$ described in Section 4.2. The experiments summarized in Table 3 were performed using sets D and E. To focus on the impact of the models on solution quality, we deactivated KaHyPar’s FM local search algorithms and only use flow-based refinement without the most balanced minimum cut heuristic. The results confirm our hypothesis that $\mathcal{F}_G$ restricts the space of possible solutions. For all flow problem sizes and all imbalances tested, $\mathcal{F}_H$ yields better solution quality. As expected, the effects are most pronounced for small flow problems and small imbalances where many vertices are likely to be border nodes. Since these nodes are locked inside their respective block in $\mathcal{F}_G$, they prevent all non-cut border nets from becoming part of the cut-set. Our model, on the other hand, allows all min-cuts that yield a feasible solution for the original partitioning problem. The fact that this effect also occurs for the graphs of set D indicates that our model can also be effective for traditional graph partitioning.

### 5.3 Configuring the Algorithm

We now evaluate different configurations of our refinement framework using $\mathcal{F}_H$ on set D and optimize the connectivity metric. In the following, KaHyPar-CA [46] is used as a reference and referred to as (-F,-M,+LS), since it neither uses (F)lows nor the (M)ost balanced minimum cut heuristic and only refines partitions using the FM (L)ocal (S)earch algorithm. This basic configuration is then successively extended with specific components. The results of our experiments are summarized in Table 4 for increasing scaling parameter $\alpha'$. In configuration CONSTANT128, all components are enabled (+F,+M,+LS) and flow-based refinements are performed every 128 uncontractions. It is used as a reference point for the quality achievable using flow-based refinement.

### Table 3. Comparing the KaFFPa Flow Model $\mathcal{F}_G$ with Our Model $\mathcal{F}_H$ as Described in Section 4.2

| $\alpha'$ | $\epsilon = 1\%$ | $\epsilon = 3\%$ | $\epsilon = 5\%$ | $\epsilon = 1\%$ | $\epsilon = 3\%$ | $\epsilon = 5\%$ |
|-----------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 1         | 7.6             | 8.1             | 7.7             | 4.7             | 4.9             | 4.8             |
| 2         | 7.5             | 6.7             | 4.9             | 4.6             | 4.1             | 3.6             |
| 4         | 6.7             | 4.0             | 2.8             | 4.2             | 3.2             | 2.5             |
| 8         | 4.9             | 2.3             | 1.5             | 3.7             | 2.3             | 1.9             |
| 16        | 3.2             | 1.4             | 1.3             | 3.0             | 1.8             | 1.6             |

The table shows the average improvement of $\mathcal{F}_H$ over $\mathcal{F}_G$ (in [%]) on benchmark sets D and E.
Table 4. Quality and Running Times for Different Framework Configurations and Increasing $\alpha'$

| $\alpha'$ | (+F,−M,−LS) | (+F,+M,−LS) | (+F,−M,+LS) | (+F,+M,+LS) | CONSTANT128 |
|---|---|---|---|---|---|
| Avg. [%] | t[s] | Avg. [%] | t[s] | Avg. [%] | t[s] | Avg. [%] | t[s] |
| 1 | −6.09 | 12.9 | −5.60 | 13.4 | 0.25 | 15.0 | 0.23 | 15.2 | 0.54 | 67.9 |
| 2 | −3.19 | 15.8 | −2.07 | 16.7 | 0.59 | 17.0 | 0.73 | 17.5 | 1.11 | 140.2 |
| 4 | −1.82 | 20.4 | −0.19 | 21.9 | 0.90 | 20.4 | 1.21 | 21.5 | 1.66 | 269.6 |
| 8 | −0.85 | 28.5 | 0.98 | 30.7 | 1.24 | 26.5 | 1.71 | 28.7 | 2.20 | 512.1 |
| 16 | −0.19 | 43.3 | 1.75 | 46.7 | 1.60 | 37.5 | 2.21 | 41.3 | 2.76 | 973.9 |

Ref. (−F,−M,+LS) 6373.88 13.7

Column Avg[%] reports the quality improvement relative to the reference configuration (−F,−M,+LS).

Table 5. Comparison of Quality Improvement and Running Times Using Speedup Heuristics

| Configuration | Avg. | Min. | $t_{flow}$[s] | t[s] |
|---|---|---|---|---|
| KaHyPar-CA | 7,077.20 | 6,820.17 | - | 29.3 |
| KaHyPar-MF | 2.47% | 2.12% | 43.0 | 72.3 |
| KaHyPar-MF(S1) | 2.41% | 2.06% | 33.9 | 63.2 |
| KaHyPar-MF(S1,S2) | 2.40% | 2.05% | 28.5 | 57.8 |
| KaHyPar-MF(S1,S2,S3) | 2.41% | 2.06% | 21.2 | 50.5 |

Column $t_{flow}$[s] refers to the running time of flow-based refinement, column t[s] to the total partitioning time.

The results indicate that only using flow-based refinement (+F,−M,−LS) is inferior to FM local search in regard to both running time and solution quality. Although the quality improves with increasing flow problem size (i.e., increasing $\alpha'$), the average connectivity is still worse than the reference configuration. Enabling the most balanced minimum cut heuristic improves partitioning quality. Configuration (+F,+M,−LS) performs better than the basic configuration for $\alpha' \geq 8$. By combining flows with the FM algorithm (+F,−M,+LS), we get a configuration that improves upon the baseline even for small flow problems. However, comparing this variant with (+F,+M,−LS) for $\alpha' = 16$, we see that using large flow problems together with the most balanced minimum cut heuristic yields solutions of comparable quality. Enabling all components (+F,+M,+LS) and using large flow problems performs best. Furthermore, we see that enabling FM local search slightly improves the running time for $\alpha' \geq 8$. This can be explained by the fact that the FM algorithm already produces good cuts between the blocks such that fewer rounds of pairwise flow refinements are necessary to improve the solution. Comparing configuration (+F,+M,+LS) with CONSTANT128 shows that performing flows more often further improves solution quality at the cost of slowing down the algorithm by more than an order of magnitude. In all further experiments, we therefore use configuration (+F,+M,+LS) with $\alpha' = 16$ for KaHyPar-MF. This configuration also performed best in the effectiveness tests presented in Appendix C. While this configuration performs better than KaHyPar-CA, its running time is still more than a factor of three higher.

We therefore perform additional experiments on set B (again, optimizing connectivity) and successively enable the speedup techniques described in Section 4.3. The results are summarized in Table 5. Only executing pairwise flow refinements on blocks that lead to an improvement on previous levels (S1) reduces the running time of flow-based refinement by a factor of 1.27, while skipping flows in case of small cuts (S2) results in a further speedup of 1.19. By additionally stopping the resizing of the flow problem as early as possible (S3), we decrease the running time of flow-based improvement by a factor of two in total, while still computing solutions of comparable quality. Thus, in the comparisons with other systems, all heuristics are enabled.
2.3:18 T. Heuer et al.

Fig. 6. Performance profiles comparing the effects of flow-based refinements on the previously best KaHyPar configurations for connectivity optimization (left) and cut-net optimization (right) on benchmark set A.

Table 6. Comparing Quality Improvements and Running Times of KaHyPar Configurations Using Flows with the Previously Best Configurations on Benchmark Set A

| Objective | Algorithm     | Avg.    | Min.    | P-value | $t[s]$  |
|-----------|---------------|---------|---------|---------|----------|
| $\lambda^{-1}$ | KaHyPar-MF     | 7,749.37 | 7,514.0 | 0$^*$   | 52.42    |
|            | KaHyPar-CA     | 2.52%   | 2.26%   | —       | 30.95    |
| cut-net    | KaHyPar-R-MF   | 6,556.05 | 6364.6  | $2.407638 \times 10^{-76}$ | 54.72    |
|            | KaHyPar-R      | 0.77%   | 0.66%   | —       | 46.10    |

The column P-value reports the results of the Wilcoxon signed rank tests. A value of 0$^*$ is used to denote results that were below machine precision.

5.4 Comparing KaHyPar Configurations—Connectivity and Cut-Net Optimization

Having configured the flow-based refinement framework, we first evaluate its direct effects on different KaHyPar configurations before performing a general comparison with all state-of-the-art systems in Sections 5.5 and 5.6. For connectivity optimization, we compare the direct $k$-way partitioning algorithm KaHyPar-CA [46] to KaHyPar-MF. For cut-net optimization, we compare the recursive bisection-based partitioning algorithm KaHyPar-R [69] to KaHyPar-R-MF. Both KaHyPar-CA and KaHyPar-R were the previously best configurations for optimizing the respective metrics [46, 69]. In both cases, the MF versions only differ from the reference configurations in the additional use of the flow-based refinement framework.

Figure 6 summarizes the experiments performed on benchmark set A. Note that there are only very few cases in which the increased running time due to flow-based refinements prevented the MF variants from partitioning an instance within the time limit of 8 hours. For both metrics, both KaHyPar-MF and KaHyPar-R-MF significantly outperform their non flow-based counterparts. While KaHyPar-MF computed the best solutions for 92.9% of all instances, the effect of flow-based refinements is slightly less pronounced for KaHyPar-R-MF (71.4%).

Looking at Table 6, we see that whereas KaHyPar-MF is 2.52% better on average than KaHyPar-CA, the solutions of KaHyPar-R-MF are 0.77% better on average than those of KaHyPar-R. A Wilcoxon signed rank test, however, reveals that the difference in solution quality is statistically significant in both cases. The smaller overall improvement of KaHyPar-R-MF can be partially explained by an observation that has already been mentioned in the seminal paper of Kernighan and Lin [53]: In recursive bisection-based algorithms, a good solution for the first bisection divides the instance into two densely connected blocks and thus makes it more difficult for further bisections.
Network Flow-Based Refinement for Multilevel Hypergraph Partitioning

Fig. 7. Performance profiles (left) and running times (right) for connectivity optimization on benchmark set A. Detailed plots for specific instance classes and values of $k$ can be found in Figure 13 and Figure 14 in Appendix E and Appendix F.

to find small cuts. Thus, improved solutions on the first recursion levels do not necessarily translate to overall improved $k$-way partitions [72]. Furthermore, once a net is cut for the first time in one of the bisections, it will remain a cut net for the rest of the partitioning process, since it is not possible to remove it from the cut-set in the following bisections.

Comparing the running times in Table 6, we can see that KaHyPar-MF is less than a factor of two slower than KaHyPar-CA, whereas the slow-down of KaHyPar-R-MF is only a factor of 1.19. We thus conclude that using flow-based refinements is viable in both settings.

5.5 Comparison with other Systems—Connectivity Optimization

We now compare KaHyPar-MF to the state-of-the-art HGP systems on benchmark set A. As can be seen in Figure 7 (left), KaHyPar-MF computes the best partitions for 70% of all benchmark instances and is within a factor 1.6 of the best for all instances except two, for which it could not compute a solution within the time limit. The fact that the performance profile of KaHyPar-MF still significantly differs from the profile of KaHyPar-CA in the presence of all other competitors again supports the conclusion that the flow-based refinement framework enables KaHyPar-MF to compute solutions of very high quality. Note that although the $\rho(1)$ value of hMetis-R is larger than that of KaHyPar-CA, this does not indicate that hMetis-R is the second best partitioning algorithm. Figure 12 in Appendix D shows that KaHyPar-CA performs better than its competitors if KaHyPar-MF is excluded from the plot.

While the performance of hMetis-R is within a factor of $\tau = 1.1$ of the best algorithm for more than 75% of all instances, there are some instances for which it performs significantly worse than the best. Although less solutions of PaToH-Q and PaToH-D are within a factor $\tau = 1.1$ from the best, the performance profiles indicate that the worst quality ratios of PaToH are smaller than those of hMetis-R. Looking at infeasible solutions, we see that Zoltan-AlgD computes imbalanced solutions for around 5% of all instances and that more than 30% of all partitions computed by hMetis-K are imbalanced and, thus, infeasible. Finally, we note that the performance profile of HYPE (the only non-multilevel algorithm) is considerably worse than the profiles of the multilevel systems—being more than a factor of two worse than the best algorithm on more than 90% of all instances.

The running times of all algorithms are shown in Figure 7 (right). We see that although our flow-based refinement framework makes KaHyPar-MF slower than KaHyPar-CA, the median running time of KaHyPar-MF is still below the median running times of Zoltan-AlgD, hMetis-R, and
Fig. 8. Visualizing the tradeoff between running time and solution quality for connectivity optimization (left) and cut-net optimization (right) on benchmark set A. The values of all algorithms are relative to KaHyPar-MF. Note the log-scale on the x-axis and the cube root scale on the y-axis, which is used to reduce right skewness [20]. Individual plots can be found in Appendix G.

hMetis-K. We furthermore see that for both configurations of PaToH as well as HYPE, the median running time is more than an order of magnitude smaller than the median running time of the other multilevel systems.

Figure 8 (left) visualizes the tradeoff between solution quality and running time on a per-instance basis. The plot shows the running time of each algorithm relative to the running time of KaHyPar-MF on the x-axis and the solution quality relative to KaHyPar-MF on the y-axis. Since we plot (KaHyPar-MF/Algorithm)-1 on the y-axis, points above zero correspond to instances where the solution of the respective partitioner was better than the solution of KaHyPar-MF, while for points below zero, KaHyPar-MF produced solutions of higher quality. Note that the x-axis uses a log-scale, while the y-axis uses a cube root scale to reduce right skewness [20]. We see that if running time is more important than solution quality, both PaToH-D and PaToH-Q currently seem to be the method of choice and that, at the cost of considerably worse solutions, HYPE can be even faster than PaToH. For partitions of very high quality, KaHyPar-MF performs better than KaHyPar-CA, hMetis-R, hMetis-K, and Zoltan-AlgD—computing solutions of higher quality in a comparable amount of time for most instances. Results of the Friedman test presented in Table 7 (top) show that the improved solution quality of KaHyPar-MF is statistically significant in all cases.

5.6 Comparison with other Systems—Cut-Net Optimization

We now turn to cut-net optimization using benchmark set A. In the following, we compare KaHyPar-R-MF to KaHyPar-R, both versions of hMetis and PaToH, to Zoltan-AlgD, and to KaHyPar-MF. We include the direct \( k \)-way partitioning algorithm KaHyPar-MF in this evaluation for the following reason: While a predecessor of KaHyPar-CA already has been shown to perform better than KaHyPar-R for connectivity optimization [2], no direct \( k \)-way configuration of KaHyPar has yet been evaluated for cut-net optimization.

As can be seen in the results summarized in Figure 9 (left), KaHyPar-MF also outperforms the other partitioning algorithms when optimizing the cut-net metric. However, by computing the best solutions for 55% of the instances, the effect is less pronounced than for connectivity optimization. The recursive bisection-based algorithm KaHyPar-R-MF computes the best solutions for 28% of all instances and still performs better than the configuration that does not use the flow-based refinement framework (KaHyPar-R).
Table 7. Results of Significance Tests Comparing KaHyPar-MF with Different KaHyPar Configurations, and with Other Systems on the Full Benchmark Set for Connectivity (Top) and Cut-net Optimization (Bottom) on Benchmark Set A

| Algorithm            | Benchmark Set       | Connectivity Optimization | Cut-Net Optimization |
|----------------------|---------------------|---------------------------|-----------------------|
|                      | DAC2012 | ISPD98 | SAT14 Primal | SAT14 Literal | SAT14 Dual | SPM    |
| KaHyPar-CA           | 4.92 × 10^{-04} | 1.72 × 10^{-06} | 0^*          | 0^*          | 0^*      | 0^*    |
| hMetis-R             | 1.50 × 10^{-04} | 2.46 × 10^{-11} | 0^*          | 0^*          | 0^*      | 0^*    |
| hMetis-K             | 9.86 × 10^{-14} | 2.23 × 10^{-07} | 0^*          | 0^*          | 0^*      | 0^*    |
| PaToH-D              | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |
| PaToH-Q              | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |
| HYPE                 | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |
| Zoltan-AlgD          | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |
| KaHyPar-R            | 2.67 × 10^{-06} | 3.33 × 10^{-16} | 0^*          | 0^*          | 0^*      | 0^*    |
| KaHyPar-R-MF         | 4.62 × 10^{-04} | 1.78 × 10^{-08} | 1.64 × 10^{-06} | 0^*          | 0^*      | 0.1613042264 |
| hMetis-R             | 1.05 × 10^{-04} | 4.35 × 10^{-13} | 2.39 × 10^{-11} | 0^*          | 0^*      | 0^*    |
| hMetis-K             | 1.08 × 10^{-06} | 1.37 × 10^{-14} | 0^*          | 0^*          | 0^*      | 0^*    |
| PaToH-D              | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |
| PaToH-Q              | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |
| Zoltan-AlgD          | 0^*      | 0^*    | 0^*          | 0^*          | 0^*      | 0^*    |

We report the p-values of the Friedman test. Note that hMetis-K and Zoltan-AlgD have slight advantages in the following comparisons because we do not disqualify imbalanced partitions in the statistical analysis. The value 0^* is used to denote results that were below machine precision.

Fig. 9. Performance profiles (left) and running times (right) for cut-net optimization on benchmark set A. Detailed plots for specific instance classes and values of $k$ can be found in Appendix E and Appendix F.

is difficult for hypergraph with large nets. If large nets have multiple pins in several blocks of the $k$-way partition, FM-based algorithms are less likely to find meaningful moves because the gain of moving a single vertex to another block is likely to be zero. Indeed, the individual performance profiles for each benchmark set (see Figure 15 in Appendix E and the results of the Friedman test presented in Table 7 (bottom)) reveal that KaHyPar-MF performs better than KaHyPar-R-MF for all benchmark sets except SAT14 Dual, whose hypergraphs have a low average vertex degree and contain many large nets. Furthermore, the effects of our flow-based refinement framework are also
limited for cut-net optimization, since improving the cut around two blocks of the \( k \)-way partition does not necessarily yield a smaller \( k \)-way cut.

The performance profiles of the other partitioners yield similar conclusions as for connectivity optimization. Looking at the running times of all algorithms in Figure 9 (right), we see that while the median running times of KaHyPar-R and KaHyPar-R-MF are comparable, adding the flow-based refinement framework increases the number of outliers. However, on average, the running time of KaHyPar-R-MF and KaHyPar-MF are comparable to the running time of hMetis-R and smaller than the running times of Zoltan-AlgD, while both PaToH configurations are again more than an order of magnitude faster than all other partitioning tools. Looking at the tradeoff between solution quality and running time in Figure 8 (right), we see that KaHyPar-MF seems to be the method of choice for high-quality partitioning, and that KaHyPar-R-MF can be a viable alternative for hypergraphs with many large nets. If running times are more important than quality, again PaToH computes partitions up to three orders of magnitude faster than KaHyPar-MF and other systems.

5.7 Using KaHyPar as a Graph Partitioner—Comparison with KaFFPa

The following experiments are performed on benchmark set E and use KaFFPa’s strong social configuration, which provides the highest solution quality for web graphs and social networks. Due to the large running times of KaFFPa and the small number of instances, we did not impose a time limit in these experiments. The results summarized in Figure 10 (left) and Table 8 compare KaHyPar-MF with the current version of KaFFPa and with KaFFPa,* which uses the flow network modifications described in Section 3.2. On average, KaHyPar-MF computes solutions that are 2.6% better than KaFFPa and its performance ratios are always within a factor of \( \tau = 1.2 \) from the best.
While the improvement of KaFFPa\(^*\) over KaFFPa is small on average, it computes better solutions than KaFFPa for 104 instances and solutions of equal quality in 11 out of the 147 cases. A Friedman test reveals that when comparing both KaFFPa and KaFFPa\(^*\) with KaHyPar-MF, the difference in solution quality is only statistically significant for the former, which demonstrates that our modifications are indeed effective for traditional graph partitioning.

Looking at the running times of all algorithms, we see that KaHyPar-MF is more than a factor of two faster than both KaFFPa configurations. This is surprising, since hypergraph partitioning inherently involves certain overheads such as generalized data structures and more complex coarsening and local search implementations.

5.8 Integrating Network Flow-Based Refinement into KaHyPar-E

The memetic algorithm KaHyPar-E employs an evolutionary framework to explore the global solution space using recombination and mutation operators specifically tailored to hypergraph partitioning. The local solution space is exploited via \(n\)-level partitioning using KaHyPar-CA. The experiments presented in this section are concerned with connectivity optimization and reuse the results from Ref. [6] for PaToH-D, hMetis-R, hMetis-K, and KaHyPar-E.\(^9\) New experiments for KaHyPar-E-MF and PaToH-Q are performed on the same machine using the same experimental setup as described in Ref. [6]: Each algorithm is given 8 hours time to partition each test instance and we perform five repetitions with different seeds. While non-evolutionary algorithms repeatedly partition each instance until the time limit is reached, KaHyPar-E and KaHyPar-E-MF evolve a population of solutions. The results are summarized in Figure 10 (right) and Table 9. Since KaHyPar-MF is roughly a factor of two slower than KaHyPar-CA, KaHyPar-E-MF spends more time on improving individual solutions of the population than KaHyPar-E. Nevertheless we can see that using stronger refinement algorithms to exploit the local solution space improves the overall solution quality. While the average improvement is small, the performance of KaHyPar-E-MF is statistically better than the performance of all other algorithms. Furthermore, we note that when given a relatively large time limit for repeated executions with different seeds, the best solutions found by PaToH-Q seem to compare favorably with the best solutions of hMetis-R.

6 CONCLUSION

We generalize KaFFPa’s flow-based refinement framework [67] from graph to hypergraph partitioning. By removing low-degree hypernodes and exploiting the fact that our flow problems are built on subhypergraphs, we reduce the size of the resulting hypergraph flow networks. Furthermore, we identify shortcomings of the KaFFPa [67] approach that restrict feasible solutions and

---

\(9\) Available from: http://algo2.iti.kit.edu/schlag/gecco2018/.
introduce an improved model that overcomes these limitations. For graph partitioning, this is done by modifying edge weights of the flow network. For hypergraph partitioning, we utilize the structure of hypergraph flow networks. Lastly, we present techniques to improve the running time of the framework by a factor of two without affecting solution quality.

The resulting hypergraph partitioner KaHyPar-MF computes better solutions than all competing algorithms and KaHyPar configurations for both cut-net and connectivity optimization while having a running time comparable to that of hMetis. While PaToH computes solutions of reasonable quality up to two orders of magnitude faster than other systems and, thus, should still be considered the method of choice if speed is more important quality, our experiments indicate that KaHyPar-MF can be seen as the state-of-the-art for high-quality hypergraph partitioning. Furthermore, it compares favorably with KaFFPa for the special case of graph partitioning. Even after enhancing KaFFPa with our improved flow model, KaHyPar-MF computes solutions of comparable quality while being more than a factor of two faster than KaFFPa’s high-quality configuration. Finally, we show that KaHyPar-MF is useful as partitioning engine to exploit the local solution space in the memetic hypergraph partitioner KaHyPar-E.

APPENDIX

A OVERVIEW AND PROPERTIES OF BENCHMARK SETS

Table 10. Basic Properties of Set D

| Class       | Hypergraph         | n     | m     | p     |
|-------------|---------------------|-------|-------|-------|
| ISPD98      | ibm06               | 32,498| 34,826| 128,182|
|            | ibm07               | 45,926| 48,117| 175,639|
|            | ibm08               | 51,309| 50,513| 204,890|
|            | ibm09               | 53,395| 60,902| 222,088|
|            | ibm10               | 69,429| 75,196| 297,567|
| SAT14 Dual | 6s9                 | 100,384| 34,317| 234,228|
|            | 6s133               | 140,968| 48,215| 328,924|
|            | 6s153               | 245,440| 85,646| 572,692|
|            | dated-10-11-u       | 629,461| 141,860| 1,429,872|
|            | dated-10-17-u       | 1,070,757| 229,544| 2,471,122|
| SAT14 Literal | 6s133              | 96,430| 140,968| 328,924|
|            | 6s153               | 171,292| 245,440| 572,692|
|            | aaa10-planning-ipc5 | 107,838| 308,235| 690,466|
|            | dated-10-11-u       | 283,720| 629,461| 1,429,872|
|            | atco_enc2_opt1_05_21| 112,732| 526,872| 2,097,393|
| SAT14 Primal | 6s153              | 85,646| 245,440| 572,692|
|            | aaa10-planning-ipc5 | 53,919| 308,235| 690,466|
|            | hwmc10-timeframe    | 163,622| 488,120| 1,138,944|
|            | dated-10-11-u       | 141,860| 629,461| 1,429,872|
|            | atco_enc2_opt1_05_21| 56,533| 526,872| 2,097,393|
| SPM         | mult_dcop_01        | 25,187| 25,187| 193,276|
|            | vibrobox            | 12,328| 12,328| 342,828|
|            | RFdevice            | 74,104| 74,104| 365,580|
|            | mixtanks_new        | 29,957| 29,957| 1,995,041|
|            | laminar_duct3D      | 67,173| 67,173| 3,833,077|

The number of pins is denoted with p.
Table 11. Basic Properties of the Web Graphs and Social Networks from Ref. [61]

| Graph                    | n   | m   |
|-------------------------|-----|-----|
| p2p-Gnutella04          | 6,405 | 29,215 |
| wordassociation-2011    | 10,617 | 63,788 |
| PGPgiantcompo           | 10,680 | 24,316 |
| email-EuAll             | 16,805 | 60,260 |
| as-22july06             | 22,963 | 48,436 |
| soc-Slashdot0902        | 28,550 | 379,445 |
| loc-brightkite          | 56,739 | 212,945 |
| enron                   | 69,244 | 254,449 |
| loc-gowalla             | 196,591 | 950,327 |
| coAuthorsCiteseer       | 227,320 | 814,134 |
| wiki-Talk               | 232,314 | 1.5M  |
| citationCiteseer        | 268,495 | 1.2M  |
| coAuthorsDBLP           | 299,067 | 977,676 |
| cnr-2000                | 325,557 | 2.7M  |
| web-Google              | 356,648 | 2.1M  |
| coPapersCiteseer        | 434,102 | 16.0M |
| coPapersDBLP            | 540,486 | 15.2M |
| as-skitter              | 554,930 | 5.8M  |
| amazon-2008             | 735,323 | 3.5M  |
| eu-2005                 | 862,664 | 16.1M |
| in-2004                 | 1.3M  | 13.6M |

B EFFECTS OF USING RANDOMIZATION IN HYPE

Figure 11 compares solution quality and running time of one iteration of HYPE’s default configuration, which was also used in the corresponding publication [60], with the best results of ten repetitions of the randomized configuration. HYPE\textsubscript{rnd} uses parameters -x to set a random seed and -m truly-random for random vertex selection. Since HYPE\textsubscript{rnd} performed worse than the default configuration regarding both solution quality and running time, we use the default configuration in our experimental evaluation.

![Figure 11](https://example.com/figure11.png)

Fig. 11. Performance profiles (left) and running times (right) for connectivity optimization comparing the default configuration of HYPE with HYPE\textsubscript{rnd}, which uses randomization. Experiments are performed on benchmark set A. Each hypergraph is partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks.
C EFFECTIVENESS TESTS

We give each configuration the same time to compute a partition. For each instance \((H, k)\), we execute each configuration once and note the largest running time \(t_{H,k}\). Then each configuration gets time \(3t_{H,k}\) to compute a partition (i.e., we take the best partition out of several repeated runs). Whenever a new run of a partition would exceed the largest running time, we perform the next run with a certain probability such that the expected running time is \(3t_{H,k}\). The results of this procedure, which was initially proposed in Ref. [67], are presented in Table 12. Combinations of flows and FM local search perform better than repeated executions of the baseline configuration, with \((+F,+M,-LS)\) and \(\alpha' = 16\) performing best.

Table 12. Results of the Effectiveness Test for Different Configurations of Our Flow-based Refinement Framework for Increasing \(\alpha'\)

| Config. (+F,−M,−LS) | (+F,+M,−LS) | (+F,−M,+LS) | (+F,+M,+LS) |
|----------------------|--------------|--------------|--------------|
| \(\alpha'\) Avg. [%] | Avg. [%] | Avg. [%] | Avg. [%] |
| 1  | −6.06 | −5.52 | 0.23 | 0.24 |
| 2  | −3.15 | −2.06 | 0.55 | 0.73 |
| 4  | −1.89 | −0.19 | 0.86 | 1.20 |
| 8  | −0.87 | 0.96 | 1.20 | 1.69 |
| 16 | −0.29 | 1.66 | 1.52 | 2.17 |
| Ref. (−F,−M,+LS) | 6,377.15 |

The quality in column Avg[%] is relative to the baseline configuration.

D PERFORMANCE PROFILES OF PREVIOUS STATE-OF-THE-ART HGP SYSTEMS

Fig. 12. Performance profiles for connectivity optimization (left) and cut-net optimization (right) of previous state-of-the-art HGP systems. Experiments are performed on benchmark set A. Each hypergraph is partitioned into \(k \in \{2, 4, 8, 16, 32, 64, 128\}\) blocks.
Fig. 13. Performance profiles comparing KaHyPar-MF with KaHyPar-CA and other partitioners for connectivity optimization for different instance classes. Experiments are performed on benchmark set A. Each hypergraph is partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks.
Fig. 14. Performance profiles comparing KaHyPar-MF with KaHyPar-CA and other partitioners for connectivity optimization for different values of $k$. Experiments are performed on benchmark set A. Each hypergraph is partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks.
Fig. 15. Performance profiles comparing KaHyPar-MF with KaHyPar-R, KaHyPar-R-MF, and other partitioners for cut-net optimization for different instance classes. Experiments are performed on benchmark set A. Each hypergraph is partitioned into \( k \in \{2, 4, 8, 16, 32, 64, 128\} \) blocks.
Fig. 16. Performance profiles comparing KaHyPar-MF with KaHyPar-R, KaHyPar-R-MF, and other partitioners for cut-net optimization for different values of $k$. Experiments are performed on benchmark set A. Each hypergraph is partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks.
F DETAILED RUNNING TIMES

Fig. 17. Comparing the running times of connectivity optimization for different benchmark sets (top) and different values of $k$ (bottom). Experiments are performed on benchmark set A. Each hypergraph is partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks.
Fig. 18. Comparing the running times of cut-net optimization for different benchmark sets (top) and different values of $k$ (bottom). Experiments are performed on benchmark set A. Each hypergraph is partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks.
G  DETAILED TRADEOFF PLOTS

Fig. 19. Visualizing the tradeoff between running time and solution quality for connectivity optimization (top) and cut-net optimization (bottom). The values of all algorithms are relative to KaHyPar-MF. Note the log-scale on the x-axis and the cube root scale on the y-axis, which is used to reduce right skewness [20].
REFERENCES

[1] P. Agrawal, B. Narendran, and N. Shivakumar. 1996. Multi-way partitioning of VLSI circuits. In 9th International Conference on VLSI Design (VLSID). 393–399.

[2] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag. 2017. Engineering a direct k-way hypergraph partitioning algorithm. In 19th Workshop on Algorithm Engineering and Experiments (ALENEX). 28–42.

[3] C. J. Alpert. 1998. The ISPD’98 circuit benchmark suite. In International Symposium on Physical Design (ISPD). 80–85.

[4] C. J. Alpert, J.-H. Huang, and A. B. Kahng. 1998. Multilevel circuit partitioning. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 17, 8 (1998), 655–667.

[5] C. J. Alpert and A. B. Kahng. 1995. Recent directions in netlist partitioning: A survey. Integr. VLSI J. 19, 1–2 (1995), 1–81.

[6] R. Andre, R. Schlag, and C. Schulz. 2018. Memetic multilevel hypergraph partitioning. In Genetic and Evolutionary Computation Conference (GECCO). 347–354.

[7] C. Aykanat, B. B. Cambazoglu, and B. Uçar. 2008. Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices. J. Parallel Distrib. Comput. 68, 5 (2008), 609–625.

[8] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner (Eds.). 2013. Proc. Graph Partitioning and Graph Clustering—10th DIMACS Implementation Challenge Workshop. Contemporary Mathematics, Vol. 588. AMS.

[9] A. Belov, D. Diepold, M. Heule, and M. Järvisalo. 2014. The SAT Competition 2014. Retrieved from http://www.satcompetition.org/2014/.

[10] R. H. Bisseling, B. O. Fagginger Auer, A. N. Yzelman, T. van Leeuwen, and Ü. V. Catalyürek. 2012. Two-dimensional approaches to sparse matrix partitioning. In Combinatorial Scientific Computing. CRC Press, New York, 321–349.

[11] E. Boman, K. Devine, V. Leung, L. A. Riesen, and Ü. V. Catalyürek. 2012. Zoltan User’s Guide. Retrieved from http://www.cs.sandia.gov/Zoltan/ug_html/ug_alg_patoh.html.

[12] Y. Boykov and V. Kolmogorov. 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Trans. Pattern Anal. Mach. Intell. 26, 9 (2004), 1124–1137.

[13] T. N. Bui and C. Jones. 1993. A heuristic for reducing fill-in in sparse matrix factorization. In SIAM Conference on Parallel Processing for Scientific Computing, 445–452.

[14] A. E. Caldwell, A. B. Kahng, and I. L. Markov. 2000. Improved algorithms for hypergraph bipartitioning. In Proceedings of the Design Automation Conference (ASP-DAC). 2000. 661–666.

[15] B. Calvo and G. Santafe. 2016. scmamp: Statistical comparison of multiple algorithms in multiple problems. The R Journal. 8, 1 (2016), 248–256. https://doi.org/10.32614/RJ-2016-017

[16] Ü. V. Catalyürek and C. Aykanat. 1999. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst. 10, 7 (Jul. 1999), 673–693.

[17] Catalyürek, Ü. V. [n.d.], ISPD98 Benchmark. Retrieved from http://bmi.osu.edu/umit/PaToH/ispd98.html.

[18] J. Cong, M. Romesis, and M. Xie. 2003. Optimality, scalability and stability study of partitioning and placement algorithms. In International Symposium on Physical Design (ISPD). ACM, 88–94.

[19] J. Cong and M. Smith. 1993. A parallel bottom-up clustering algorithm with applications to circuit partitioning in VLSI design. In 30th Conference on Design Automation (DAC). 755–760.

[20] N. J. Cox. 2011. Stata tip 96: Cube roots.

[21] T. A. Davis and Y. Hu. 2011. The university of Florida sparse matrix collection.

[22] C. Curino, E. Jones, Y. Zhang, and S. Madden. 2010. Schism: A workload-driven approach to database replication and partitioning. Proceedings VLDB Endow. 3, 48–57 (Sept. 2010), 48–57.

[23] T. A. Davis and Y. Hu. 2011. The university of Florida sparse matrix collection.

[24] C. Fiduccia and R. Mattheyses. 1982. A linear time heuristic for improving network partitions. In 19th ACM/IEEE Design Automation Conf. 175–181.

[25] H. Finner. 1993. On a monotonicity problem in step-down multiple test procedures. J. Am. Stat. Assoc. 88, 423 (1993), 920–923.
[31] D. R. Ford and D. R. Fulkerson. 1962. *Flows in Networks*. Princeton University Press.

[32] L. R. Ford and D. R. Fulkerson. 1956. Maximal flow through a network. *Can. J. Math.* 8, 3 (1956), 399–404.

[33] M. Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.* 32, 200 (1937), 675–701.

[34] M. Friedman. 1940. A comparison of alternative tests of significance for the problem of m rankings. *Ann. Math. Stat.* 11, 1 (1940), 86–92.

[35] S. García, A. Fernández, J. Luengo, and F. Herrera. 2010. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Inf. Sci.* 180, 10 (2010), 2044–2064.

[36] S. García and F. Herrera. 2008. An extension on “statistical comparisons of classifiers over multiple data sets” for all pairwise comparisons. *J. Mach. Learn. Res.* 9, (Dec.2008), 2677–2694.

[37] A. Goldberg, S. Hed, H. Kaplan, R. Tarjan, and R. Werneck. 2011. Maximum flows by incremental breadth-first search. *19th European Symposium on Algorithms (ESA)* (2011), 457–468.

[38] A. V. Goldberg and R. E. Tarjan. 1988. A new approach to the maximum-flow problem. *J. ACM* 35, 4 (1988), 921–940.

[39] N. Gould and J. Scott. 2016. A note on performance profiles for benchmarking software. *ACM Trans. Math. Software* 43, 2 (2016), 15:1–15:5.

[40] S. Hauck and G. Borriello. 1997. An evaluation of bipartitioning techniques. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 16, 8 (1997), 849–866.

[41] B. Heintz and A. Chandra. 2014. Beyond graphs: Toward scalable hypergraph analysis systems. *ACM SIGMETRICS Performance Evaluation Review* 41, 4 (April 2014), 94–97.

[42] B. Hendrickson and R. Leland. 1995. A multi-level algorithm for partitioning graphs. *1995 Conference on Supercomputing (SC)* 0 (1995), 28.

[43] T. Heuer. 2018. *High Quality Hypergraph Partitioning via Max-Flow-Min-Cut Computations*. Master’s Thesis. KIT.

[44] T. Heuer, P. Sanders, and S. Schlag. 2018. Network flow-based refinement for multilevel hypergraph partitioning. *CoRR* abs/1802.03587 (2018). arxiv:1802.03587

[45] T. Heuer, P. Sanders, and S. Schlag. 2018. Network flow-based refinement for multilevel hypergraph partitioning. In *17th International Symposium on Experimental Algorithms (SEA)*, Vol. 103. 1:1–1:19.

[46] T. Heuer and S. Schlag. 2017. Improving coarsening schemes for hypergraph partitioning by exploiting community structure. In *16th International Symposium on Experimental Algorithms (SEA)*, 21:1–21:19.

[47] T. Hothorn, K. Hornik, van de Wiel, M. A., and A. Zeileis. 2006. A lego system for conditional inference. *The American Statistician* 60, 3 (2006), 257–263.

[48] T. C. Hu and K. Moerder. 1985. Multiterminal flows in a hypergraph. In *VLSI Circuit Layout: Theory and Design*, T.C. Hu and E.S. Kuh (Eds.). IEEE Press, Chapter 3, 87–93.

[49] R. L. Iman and J. M. Davenport. 1980. Approximations of the critical region of the friedman statistic. *Commun. Stat. Theor. Methods* 9, 6 (1980), 571–595.

[50] I. Kabljo, B. Karrer, M. Pundir, S. Pupyrev, A. Shalita, Y. Akhremtsev, and Presta. A. 2017. Social hash partitioner: A scalable distributed hypergraph partitioner. *PVldb* 10, 11 (2017), 1418–1429.

[51] G. Karypis and V. Kumar. 1999. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Trans. Very Large Scale Integr VLSI Syst.* 7, 1 (1999), 69–79.

[52] G. Karypis and V. Kumar. 1999. Multilevel K-way hypergraph partitioning. In *36th Design Automation Conference (DAC)*. ACM, 343–348.

[53] B. W. Kernighan and S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J.* 49, 2 (Feb. 1970), 291–307.

[54] E. Lawler. 1973. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Whinston.

[55] T. Lengauer. 1990. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc.

[56] E. Lawler. 1976. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Whinston.

[57] T. Lengauer. 1995. *International POS-14.5th Pragmatics of SAT Workshop (EPiCSeriesin Combinatorial Optimization)*. John Wiley & Sons, Inc.

[58] Z. Mann and P. Papp. 2014. Formula partitioning revisited. In *POS-14. 5th Pragmatics of SAT Workshop (EPiC Series in Computing)*, Daniel Le Berre (Ed.), Vol. 27. EasyChair, 41–56.

[59] C. Mayer, R. Mayer, S. Bhowmik, L. Epplle, and K. Rothermel. 2018. HYPE: Massive hypergraph partitioning with neighborhood expansion. In *IEEE International Conference on Big Data*. 458–467.

[60] H. Meyerhenke, P. Sanders, and C. Schulz. 2014. Partitioning complex networks via size-constrained clustering. In *13th International Symposium on Experimental Algorithms (SEA)*. 351–363.
[62] D. A. Papa and I. L. Markov. 2007. Hypergraph partitioning and clustering. In Handbook of Approximation Algorithms and Metaheuristics. T. F. González (Ed.). Chapman and Hall/CRC.

[63] J.-C. Picard and M. Queyranne. 1980. On the structure of all minimum cuts in a network and applications. Combinatorial Optimization II (1980), 8–16.

[64] J. Pistorius and M. Minoux. 2003. An improved direct labeling method for the max–flow min–cut computation in large hypergraphs and applications. Int. Trans. Oper. Res. 10, 1 (2003), 1–11.

[65] S. Riyavong. 2003. Experiments on Sparse Matrix Partitioning. Technical Report CERFACS Working Note WN/PA/03/32. CERFACS. 20 pages.

[66] L. A. Sanchis. 1989. Multiple-way network partitioning. IEEE Trans. Comput. 38, 1 (1989), 62–81.

[67] P. Sanders and C. Schulz. 2011. Engineering multilevel graph partitioning algorithms. In 19th European Symposium on Algorithms (ESA) (LNCS), Vol. 6942. Springer, 469–480.

[68] P. Sanders and C. Schulz. 2013. Think locally, act globally: Highly balanced graph partitioning. In Proceedings of the 12th International Symposium on Experimental Algorithms (SEA’13) (LNCS), Vol. 7933. Springer, 164–175.

[69] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. 2016. k-way hypergraph partitioning via n-level recursive bisection. In 18th Workshop on Algorithm Engineering and Experiments (ALENEX). 53–67.

[70] D. G. Schweikert and B. W. Kernighan. 1972. A proper model for the partitioning of electrical circuits. In 9th Design Automation Workshop (DAC). ACM, 57–62.

[71] R. Shaydulin and I. Safro. 2018. Aggregative coarsening for multilevel hypergraph partitioning. In 17th International Symposium on Experimental Algorithms (SEA), Vol. 103. 2:1–2:15.

[72] H. D. Simon and S.-H. Teng. 1997. How good is recursive bisection? SIAM J. Sci. Comput. 18, 5 (Sept. 1997), 1436–1445.

[73] A. Trifunovic. 2006. Parallel Algorithms for Hypergraph Partitioning. Ph.D. Dissertation. University of London.

[74] A. Trifunovi and W. J. Knottenbelt. 2008. Parallel multilevel algorithms for hypergraph partitioning. J. Parallel Distrib. Comput. 68, 5 (2008), 563–581.

[75] Ü. V. Çatalyürek, M. Deveci, K. Kaya, and B. Uçar. 2012. UMPAs: A multi-objective, multi-level partitioner for communication minimization. See Ref. [8], 53–66.

[76] B. Uçar and C. Aykanat. 2004. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM Journal on Scientific Computing 25, 6 (2004), 1837–1859.

[77] B. Vastenhouw and R. H. Bisseling. 2005. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Rev. 47, 1 (2005), 67–95.

[78] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. 2012. The DAC 2012 routability-driven placement contest and benchmark suite. In 49th Annual Design Automation Conference (DAC). ACM, 774–782.

[79] F. Wilcoxon. 1945. Individual comparisons by ranking methods. Biometrics Bulletin 1, 6 (1945), 80–83.

[80] H. H. Yang and D. F. Wong. 1996. Efficient network flow based min-cut balanced partitioning. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 15, 12 (1996), 1533–1540.

[81] C. Zhang, F. Wei, Q. Liu, Z. G. Tang, and Li. Z. 2017. Graph edge partitioning via neighborhood heuristic. In 23rd International Conference on Knowledge Discovery and Data Mining (KDD). 605–614.

Received October 2018; revised March 2019; accepted May 2019