Network Flow-Based Refinement for Multilevel Hypergraph Partitioning

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

Peter Sanders
Karlsruhe Institute of Technology, Germany
sanders@kit.edu

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

Abstract

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 KaFFPa from graphs to hypergraphs and is integrated into the hypergraph partitioner 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 while still having a running time comparable to that of hMetis.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Multilevel Hypergraph Partitioning, Network Flows, Refinement

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.1

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.03587`.

1 Introduction

Given an undirected hypergraph $H = (V, E)$, the *k-way hypergraph partitioning problem* is to partition the vertex set into k disjoint blocks of bounded size (at most $1 + \varepsilon$ times the average block size) such that an objective function involving the cut hyperedges is minimized. Hypergraph partitioning (HGP) has many important applications in practice such as scientific computing [10] or VLSI design [40]. Particularly VLSI design is a field where small improvements can lead to significant savings [53].

It is well known that HGP is NP-hard [35], which is why practical applications mostly use heuristic *multilevel* algorithms [9, 11, 22, 23]. 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. *All* state-of-the-art HGP algorithms [2, 4, 6, 14, 25, 29, 30, 31, 45, 48, 49, 51] either use variations of the Kernighan-Lin (KL) [32, 46] or the Fiduccia-Mattheyses (FM) heuristic [17, 43], or simpler greedy algorithms [30, 31] 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 [31]. 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 [50]. Thus the gain of moving a single vertex to another block is likely to be *zero* [38].

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 [19]. 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 [37, 54]. Sanders and Schulz [44] presented a max-flow-based improvement algorithm for graph partitioning which is integrated into the multilevel partitioner KaFFPa and computes high quality solutions.

Outline and Contribution. Motivated by the results of Sanders and Schulz [44], we generalize the max-flow min-cut refinement framework of KaFFPa from graphs to hypergraphs. After introducing basic notation and giving a brief overview of related work and the techniques used in KaFFPa in Section 2, we explain how hypergraphs are transformed into flow networks and present a technique to reduce the size of the resulting hypergraph flow network in Section 3.1. In Section 3.2 we then show 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 the solution quality of a k-way partition. We furthermore identify shortcomings of the KaFFPa approach that restrict the search space of feasible solutions significantly and introduce an advanced model that overcomes these limitations by exploiting the structure of hypergraph flow networks. We implemented our algorithm in the open source HGP framework KaHyPar and therefore briefly discuss implementation details and techniques to improve the running time in Section 3.3. Extensive experiments presented in Section 4 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 refinement significantly improves partitioning quality. The resulting 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. On a large benchmark set consisting of 3222 instances from various application domains, KaHyPar-MF computes the best partitions in 2427 cases.

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 $c : V \to \mathbb{R}_{>0}$ and net weights $\omega : E \to \mathbb{R}_{>0}$, 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 ω to sets, i.e., $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. A vertex v is *incident* to a net e if $v \in e$. $\mathrm{I}(v)$ denotes the set of all incident nets of v. The *degree* of a vertex v is $d(v) := |\mathrm{I}(v)|$. The *size* $|e|$ of a net e is the number of its pins. Given a subset $V' \subset V$, the *subhypergraph* $H_{V'}$ is defined as $H_{V'} := (V', \{e \cap V' \mid e \in E : e \cap V' \neq \emptyset\})$.

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 Π *ε-balanced* if each block $V_i \in \Pi$ satisfies the *balance constraint*: $c(V_i) \leq L_{\max} := (1+\varepsilon)\lceil \frac{c(V)}{k} \rceil$ for some parameter ε. 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 $\lambda(e) > 1$. Given a k-way partition Π 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 ε-balanced k-way partition Π of a hypergraph H that minimizes an objective function over the cut nets for some ε. 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 [15]. In this paper, we use the $(\lambda - 1)$-metric. Optimizing both objective functions is known to be NP-hard [35]. In the following, we use *nodes* and *edges* when referring to graphs and *vertices* and *nets* when referring to hypergraphs. Hypergraphs can be represented as *bipartite* graphs [27]: In the *star-expansion* $G_*(V \dot\cup E, F)$ the vertices and nets of H form the node set and for each net $e \in \text{I}(v)$, we add an edge (e, v) to G_*. The edge set F is thus defined as $F := \{(e, v) \mid e \in E, v \in e\}$. Each net in E therefore corresponds to a *star* in G_*.

Let $G = (V, E, c, \omega)$ be a weighted directed graph. We use the same notation to refer to node weights c, edge weights ω, and node degrees $d(v)$. Furthermore $\Gamma(\text{u}) := \{v : (u, v) \in E\}$ denotes the neighbors of node u. A *path* $P = \langle v_1, \ldots, v_k \rangle$ is a sequence of nodes, such that each pair of consecutive nodes is connected by an 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 $v \in B$. A subset $S \subset V$ 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 $\mathcal{N} = (\mathcal{V}, \mathcal{E}, c)$ 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 $f : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that satisfies the *capacity constraint* $\forall u, v \in \mathcal{V} : f(u, v) \leq c(u, v)$, the *skew symmetry constraint* $\forall u, v \in \mathcal{V} \times \mathcal{V} : f(u, v) = -f(v, u)$, and the *flow conservation constraint* $\forall u \in \mathcal{V} \setminus \{s, t\} : \sum_{v \in \mathcal{V}} f(u, v) = 0$. The value of a flow $|f| := \sum_{v \in \mathcal{V}} f(s, v)$ is defined as the total amount of flow transferred from s to t. The *residual capacity* is defined as $r_f(u, v) = c(u, v) - f(u, v)$. 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 $\mathcal{N}$ 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 [19].

2.2 Related Work

Hypergraph Partitioning. HGP has evolved into a broad research area since the 1990s. We refer to existing literature [5, 7, 40, 47] for an extensive overview. Well-known multilevel HGP software packages with certain distinguishing characteristics include PaToH [10] (originating from scientific computing), hMetis [30, 31] (originating from VLSI design), KaHyPar [2, 25, 45] (general purpose, n-level), Mondriaan [51] (sparse matrix partitioning), MLPart [4] (circuit partitioning), Zoltan [14], Parkway [48] and SHP [29] (distributed), UMPa [49] (directed hypergraph model, multi-objective), and kPaToH (multiple constraints, fixed vertices) [6]. All of these tools use move-based local search algorithms in the refinement phase.

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 [33], who showed that it can be reduced to computing maximum flows in directed graphs. Hu and Moerder [27] 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 [34]. Yang and Wong [54] use repeated, incremental max-flow min-cut computations on the Lawler network [33] to find ε-balanced hypergraph bipartitions. Solution quality and running time of this algorithm are improved by Lillis and Cheng [36] by introducing advanced heuristics to select source and sink nodes. Furthermore, they present a preflow-based [20] min-cut algorithm that implicitly operates on the star-expanded hypergraph. Pistorius and Minoux [42] generalize the algorithm of Edmonds and Karp [16] to hypergraphs by labeling both vertices and nets. Liu and Wong [37] simplify Lawler's hypergraph flow network [33] 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. Note that the above-mentioned approaches to model hypergraphs as flow networks for max-flow min-cut computations do not contradict the negative results of Ihler et al. [28], who show that, *in general*, there does not exist an edge-weighted graph $G = (V, E)$ that correctly represents the min-cut properties of the corresponding hypergraph $H = (V, E)$.

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 $\mathcal{O}(\log n)$ 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 [17, 43] are used to refine the solution. Our work builds on KaHyPar-CA [25], which is a direct k-way partitioning algorithm for optimizing the $(\lambda - 1)$-metric. It uses an improved coarsening scheme that incorporates global information about the community structure of the hypergraph into the coarsening process.

2.3 The Flow-Based Improvement Framework of KaFFPa

We discuss the framework of Sanders and Schulz [44] in greater detail, since our work makes use of the techniques proposed by the authors. For simplicity, we assume $k = 2$. The techniques can be applied on a k-way partition by repeatedly executing the algorithm on pairs of adjacent blocks. To schedule 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.

An ε-balanced bipartition of a graph $G = (V, E, c, \omega)$ is improved with flow computations as follows. The basic idea is to construct a flow network $\mathcal{N}$ based on the induced subgraph $G[B]$, where $B \subseteq V$ is a set of nodes around the cut of G. The size of B is controlled by an imbalance factor $\varepsilon' := \alpha\varepsilon$, where α is a scaling parameter that is chosen adaptively depending on the result of the min-cut computation. If the heuristic found an ε-balanced partition using ε', the cut is accepted and α is increased to $\min(2\alpha, \alpha')$ where α' 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 $B := B_1 \cup B_2$ is constructed by performing two breadth-first searches (BFS). The first BFS is done in the induced subgraph $G[V_1]$. It is initialized with the boundary nodes of V_1 and stops if $c(B_1)$ would exceed $(1+\varepsilon')\lceil \frac{c(V)}{2} \rceil - c(V_2)$. The second BFS constructs B_2 in an analogous fashion using $G[V_2]$. Let $\delta B := \{u \in B \mid \exists (u,v) \in E : v \notin B\}$ be the border of B. Then $\mathcal{N}$ is constructed by connecting all border nodes $\delta B \cap V_1$ of $G[B]$ to the source s and all border nodes $\delta B \cap V_2$ to the sink t using directed edges with an edge weight of ∞. 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. For $\alpha = 1$, the size of B thus ensures that the flow network $\mathcal{N}$ has the *cut property*, i.e., each (s,t)-min-cut in $\mathcal{N}$ yields an ε-balanced partition of G with a possibly smaller cut. For larger values of α, this does not have to be the case.

After computing a max-flow in $\mathcal{N}$, 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 information about *all* (s,t)-min-cuts [41]. 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 $\mathcal{N}_f$ [41]. 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 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 as bipartition.

3 Hypergraph Max-Flow Min-Cut Refinement

In the following, we generalize the KaFFPa algorithm to hypergraph partitioning. In Section 3.1 we first show how hypergraph flow networks $\mathcal{N}$ 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 3.2 then explains how $\mathcal{N}$ is used to build a flow problem $\mathcal{F}$ based on the subhypergraph $H_B = (V_B, E_B)$. $\mathcal{F}$ is constructed such that an (s,t)-max-flow computation optimizes the *cut* metric of the bipartition $\Pi_2 = (V_i, V_j)$ of H_B and thus improves the $(\lambda - 1)$-metric in H. Section 3.3 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 Appendix A.

3.1 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 [27].

This network is then transformed into the *edge-capacitated* flow network $\mathcal{N} = (\mathcal{V}, \mathcal{E}, c)$ of Lawler [33] as follows: $\mathcal{V}$ contains all non-star nodes v. For each star-node e, add two *bridging* nodes e' and e'' to $\mathcal{V}$ and a *bridging* edge (e', e'') with capacity $c(e', e'') = c(e)$ to $\mathcal{E}$. For each neighbor $u \in \Gamma(e)$, add two edges (u, e') and (e'', u) with infinite capacity to $\mathcal{E}$.

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 [37] 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

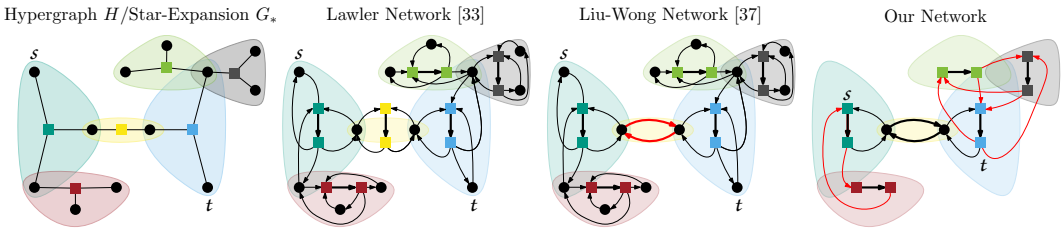

Figure 1 Unweighted hypergraph H with overlayed star-expansion G_* and illustration of the corresponding hypergraph flow networks. Thin, directed edges have infinite capacity, thick edges have unit capacity. Differences between the networks are highlighted in red: Special treatment of two-pin nets in the network of Liu and Wong [37], removal of low degree vertices in our network.

capacity $c(e)$. For each such star-node, this decreases the network size by two nodes and three edges. Figure 1 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 [33], 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 [18]. We then apply transformation of Liu and Wong.

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 $\mathcal{N}_f$ starting from s [33]. 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 1.** *Let f be a maximum (s,t)-flow in the Lawler network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ of a hypergraph $H = (V, E)$ and $(\mathcal{S}, \mathcal{V} \setminus \mathcal{S})$ be the corresponding (s,t)-min-cut in $\mathcal{N}$. Then for each node $v \in \mathcal{S} \cap V$, the residual graph $\mathcal{N}_f = (\mathcal{V}_f, \mathcal{E}_f)$ contains at least one path $\langle s, \ldots, e'' \rangle$ to a bridging node e'' of a net $e \in \mathrm{I}(v)$.*

Proof. The proof can be found in Appendix B. ◀

Thus $(A, V \setminus A)$ is an (s,t)-min-cut of H, where $A := \{v \in e \mid \exists e \in E : \langle s, \ldots, e'' \rangle \text{ in } \mathcal{N}_f\}$. Furthermore this allows us to employ the most balanced minimum cut heuristic as described in Section 2.3. By the definition of closed sets it follows that if a bridging node e'' is contained in a closed set C, then all nodes $v \in \Gamma(e'')$ (which correspond 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.

3.2 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 *internal* border nodes $\overrightarrow{B} := \{v \in V_B \mid \exists e \in E : \{u, v\} \subseteq e \wedge u \notin V_B\}$ and the set of *external* border nodes $\overleftarrow{B} := \{u \notin V_B \mid \exists e \in E : \{u, v\} \subseteq e \wedge v \in V_B\}$. Similarly, we distinguish between *external* nets ($e \cap V_B = \emptyset$) with no pins inside H_B, *internal* nets ($e \cap V_B = e$) with all pins inside H_B, and the set of *border* nets $\overleftrightarrow{E_B} := \{e \in E \mid e \in \mathrm{I}(\overrightarrow{B}) \cap \mathrm{I}(\overleftarrow{B})\}$.

A hypergraph flow problem consists of a flow network $\mathcal{N}_B = (\mathcal{V}_B, \mathcal{E}_B)$ derived from H_B and two *additional* nodes s and t that are connected to some nodes $v \in \mathcal{V}_B$. It has the cut property if the max-flow induced min-cut bipartition Π_f of H_B does not increase the $(\lambda - 1)$-metric in H. Thus it has to hold that $\mathrm{cut}(\Pi_f) \leq \mathrm{cut}(\Pi_2)$. While external nets are not affected by a max-flow computation, the max-flow min-cut theorem [19] ensures the cut property for all internal nets. Border nets however require special attention. Since a border net e is only *partially* contained in H_B and $\mathcal{N}_B$, it will remain connected to the blocks of its external border nodes in H. In case external border nodes connect e to both V_i and V_j, it will remain a cut net in H even if it is removed from the cut-set in Π_f. It is therefore necessary to "encode" information about external border nodes into the flow problem.

The KaFFPa Model and its Limitations. In KaFFPa, this is done by directly connecting internal border nodes $\overrightarrow{B}$ to s and t. This approach can also be used for hypergraphs. In the hypergraph flow problem $\mathcal{F}_G$, the source s is connected to all nodes $\mathcal{S} = \overrightarrow{B} \cap V_i$ and all nodes $\mathcal{T} = \overrightarrow{B} \cap V_j$ are connected to t using directed edges with infinite capacity. While this ensures that $\mathcal{F}_G$ has the cut property, it unnecessarily *restricts* the search space. Since all internal border nodes $\overrightarrow{B}$ are connected to either s or t, *every* min-cut $(S, V_B \setminus S)$ will have $\mathcal{S} \subseteq S$ and $\mathcal{T} \subseteq V_B \setminus S$. The KaFFPa model therefore prevents *all* min-cuts in which any non-cut border net (i.e., $e \in \overleftrightarrow{E_B}$ with $\lambda(e) = 1$) becomes part of the cut-set. This restricts the space of possible solutions, since corridor B was computed such that *even* a min-cut along either side of the border would result in a feasible cut in H_B. Thus, ideally, *all* vertices $v \in B$ should be able to change their block as result of an (s, t)-max-flow computation on $\mathcal{F}_G$ – not only vertices $v \in B \setminus \overrightarrow{B}$. This limitation becomes increasingly relevant for hypergraphs with large nets as well as for partitioning problems with small imbalance ε, since large nets are likely to be only partially contained in H_B and tight balance constraints enforce small B-corridors. While the former is a problem only for HGP, the latter also applies to GP.

A more flexible Model. We exploit the structure of hypergraph flow networks such that (s, t)-max-flow computations can also cut through non-cut border nets. Instead of directly connecting s and t to internal border nodes $\overrightarrow{B}$ and thus preventing all min-cuts in which these nodes switch blocks, we conceptually extend H_B to contain all external border nodes $\overleftarrow{B}$ and all border nets $\overleftrightarrow{E_B}$. The resulting hypergraph is $\overleftarrow{H_B} = (V_B \cup \overleftarrow{B}, \{e \in E \mid e \cap V_B \neq \emptyset\})$. The key insight now is that by using the flow network of $\overleftarrow{H_B}$ and connecting s resp. t to the *external* border nodes $\overleftarrow{B} \cap V_i$ resp. $\overleftarrow{B} \cap V_j$, 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 [19], this flow problem has the cut property, since all border nets of H_B are now internal nets and all external border nodes $\overleftarrow{B}$ are locked inside their block. However, it is not necessary to use $\overleftarrow{H_B}$ instead of H_B to achieve this result. For all $v \in \overleftarrow{B}$ the flow network of $\overleftarrow{H_B}$ contains paths $\langle s, v, e' \rangle$ and $\langle e'', v, t \rangle$ that only involve infinite-capacity edges. Therefore we can remove all nodes $v \in \overleftarrow{B}$ by directly connecting s and t to the

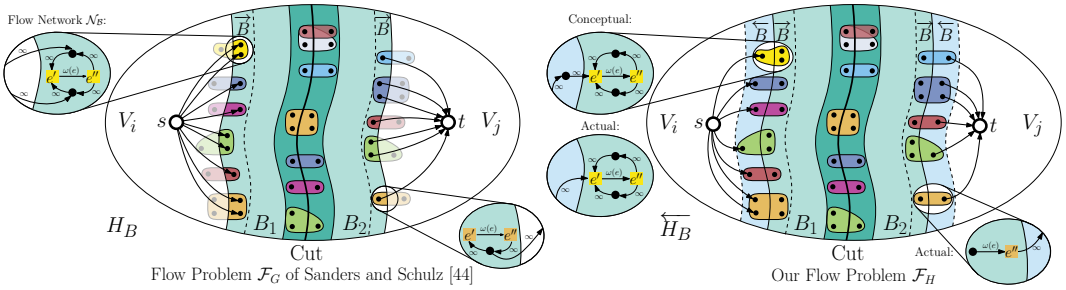

Figure 2 Comparison of the KaFFPa flow problem $\mathcal{F}_G$ and our flow problem $\mathcal{F}_H$. For clarity the zoomed in view is based on the Lawler network.

corresponding *bridging nodes* e', e'' via infinite-capacity edges without affecting the maximal flow [18]. More precisely, in the flow problem $\mathcal{F}_H$, we connect s to all bridging nodes e' corresponding to border nets $e \in \overleftrightarrow{E_B} : e \subset \overleftarrow{B} \cap V_i$ and all bridging nodes e'' corresponding to border nets $e \in \overleftrightarrow{E_B} : e \subset \overrightarrow{B} \cap V_j$ to t using directed, infinite-capacity edges.

Single-Pin Border Nets. Furthermore, we model border nets with $|e \cap \overrightarrow{B}| = 1$ more efficiently. For such a net e, the flow problem contains paths of the form $\langle s, e', e'', v\rangle$ or $\langle v, e', e'', t\rangle$ which can be replaced by paths of the form $\langle s, e', v\rangle$ or $\langle v, e'', t\rangle$ 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 $\mathcal{F}_G$ and $\mathcal{F}_H$ is shown in Figure 2.

3.3 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 recalculations, it therefore uses a cache to maintain the gain values of FM moves throughout the n-level hierarchy [2]. In order to combine 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 blocks lead to an improvement on previous levels (S1). We skip flow-based refinement if the cut between two adjacent blocks is less than ten 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).

4 Experimental Evaluation

We implemented the max-flow min-cut refinement algorithm in KaHyPar. Our implementation is available from `http://www.kahypar.org`. The code is written in C++ and compiled using g++-5.2 with flags `-O3 -march=native`. We refer to our new algorithm as KaHyPar-MF.

Instances. All experiments use the benchmark set of Heuer and Schlag [25], which contains 488 hypergraphs derived from four benchmark sets: the ISPD98 VLSI Circuit Benchmark Suite [3], the DAC 2012 Routability-Driven Placement Contest [52], the SuiteSparse Matrix Collection [13], and the international SAT Competition 2014 [8]. All hypergraphs have unit vertex and net weights. The full benchmark set is referred to as set A. We furthermore use the representative subset of 165 hypergraphs proposed in [25] (set B) and a smaller subset consisting of 25 hypergraphs (set C). Furthermore we use 15 graphs from [39] to compare our flow model $\mathcal{F}_H$ to the KaFFPa model $\mathcal{F}_G$ (set D). Unless mentioned otherwise, all (hyper-)graphs are partitioned into $k \in \{2, 4, 8, 16, 32, 64, 128\}$ blocks with $\varepsilon = 0.03$. For each value of k, a k-way partition is considered to be *one* test instance, resulting in a total of 105 instances for set D, 175 instances for set C, 1155 instances for set B and 3416 instances for set A. An overview about the different benchmark sets is given in Appendix C.

System and Methodology. We compare KaHyPar-MF to KaHyPar-CA, the k-way (hMetis-K) and recursive bisection variants (hMetis-R) of hMetis 2.0 (p1) [30, 31], and PaToH 3.2 [10], as these provide the best solution quality [2, 25]. The results of these tools are available online[1] and report the *arithmetic mean* of the computed cut and running time as well as the best cut found for ten repetitions with different seeds per instance. Since PaToH ignores the seed if configured to use the quality preset, the results contain one run of the quality (PaToH-Q) and ten runs using the default preset (PaToH-D). Each partitioner had a time limit of eight hours per instance. We use the same number of repetitions and the same time limit for our experiments[2]. Furthermore the experiments are performed on the same machine, which runs RHEL 7.2 and consists of two Intel Xeon E5-2670 Octa-Core processors clocked at 2.6 GHz, 64 GB main memory, 20 MB L3- and 8x256 KB L2-Cache. When averaging over different instances we use the *geometric mean* in order to give every instance a comparable influence on the final result. To compare the algorithms in terms of solution quality, we use *improvement plots*. For each algorithm, these plots relate the minimum connectivity of KaHyPar-MF to the minimum connectivity of the algorithm on a *per-instance* basis. For each algorithm, these ratios are sorted in decreasing order. The plots use a cube root scale for the y-axis to reduce right skewness [12] and show the improvement of KaHyPar-MF in percent (i.e., $1-$(KaHyPar-MF/algorithm)). A value below zero indicates that KaHyPar-MF performed worse than the corresponding algorithm, while a value above zero indicates that KaHyPar-MF performed better than the algorithm in question. A value of zero implies that the partitions of both algorithms had the same solution quality. Values above one correspond to infeasible solutions that violated the balance constraint. To include instances with a connectivity of zero into the results, we set these values to *one* for ratio computations.

4.1 Evaluating Flow Networks and Models

Flow Networks. To analyze the effects of the different flow networks we compute five bipartitions for each hypergraph of set B with KaHyPar-CA. These are then used to generate flow networks for a corridor of size $|B| = 25\,000$ vertices around the cut, from which we create flow problems $\mathcal{F}_H$. Due to space constraints, we only report how the reductions in network size translate to improved running times of the max-flow algorithm and refer to the full version of the paper [26] for a detailed discussion. We use the highly tuned IBFS

[1] `http://algo2.iti.kit.edu/schlag/sea2017/`
[2] Detailed per-instance results can be found online: `http://algo2.iti.kit.edu/schlag/sea2018/`

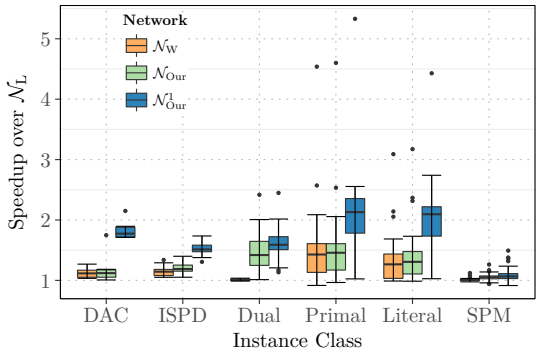

Figure 3 Speedup of the IBFS [21] max-flow algorithm over the execution on $\mathcal{N}_L$.

Table 1 Comparing the KaFFPa flow model $\mathcal{F}_G$ with our model $\mathcal{F}_H$ as described in Section 3.2. The table shows the average improvement of $\mathcal{F}_H$ over $\mathcal{F}_G$ (in [%]) on benchmark sets C and D.

	Hypergraphs (Set C)			Graphs (Set D)		
α'	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$	$\varepsilon = 1\%$	$\varepsilon = 3\%$	$\varepsilon = 5\%$
1	7.7	8.1	7.6	11.7	11.3	10.5
2	7.9	6.6	4.8	11.0	9.1	7.8
4	6.9	3.9	2.7	9.9	7.3	5.4
8	5.1	2.3	1.5	8.6	5.3	3.9
16	3.4	1.3	1.2	7.0	4.1	3.5

algorithm [21] to compute min-cuts, which performed best in preliminary experiments [24, 26]. Figure 3 compares the speedups of IBFS when executed on the Liu-Wong network $\mathcal{N}_W$, our network $\mathcal{N}_{\text{Our}}$, and $\mathcal{N}^1_{\text{Our}}$ to the execution on the Lawler network $\mathcal{N}_L$. $\mathcal{N}^1_{\text{Our}}$ exploits the fact that our flow problems are based on subhypergraphs H_B by additionally modeling single-pin border nets more efficiently as described in Section 3.2. We see that the algorithm benefits from improved network models on all instance classes except SPM. These instances 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 $\mathcal{N}_W$ speeds up computation for Primal and Literal instances, $\mathcal{N}_{\text{Our}}$ additionally leads to a speedup for Dual instances. Using $\mathcal{N}^1_{\text{Our}}$, which combines these techniques with efficient border net modeling results in an average speedup between 1.52 and 2.21 (except for SPM instances). Therefore we use $\mathcal{N}^1_{\text{Our}}$ in all following experiments.

Flow Models. We now compare the flow model $\mathcal{F}_G$ of KaFFPa to our advanced model $\mathcal{F}_H$ described in Section 3.2. The experiments summarized in Table 1 were performed using benchmark sets C and D. 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 search 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. In all following experiments, we use $\mathcal{F}_H$.

Table 2 Quality and running times for different framework configurations and increasing α'. Column Avg[%] reports the quality improvement relative to the reference configuration (-F,-M,+FM).

	(+F,-M,-FM)		(+F,+M,-FM)		(+F,-M,+FM)		(+F,+M,+FM)		Constant128	
α'	Avg[%]	$t[s]$	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,+FM)		6373.88	13.7						

Table 3 Comparison of quality improvement and running times using speedup heuristics. Column $t_{\text{flow}}[s]$ refers to the running time of flow-based refinement, $t[s]$ to the total partitioning time.

Configuration	Avg	Min	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	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

4.2 Configuring the Algorithm

We now evaluate different configurations of our refinement framework on set C. In the following, KaHyPar-CA [25] is used as a reference and referred to as (-F,-M,+FM), since it neither uses (F)lows nor the (M)ost balanced minimum cut heuristic and only relies on the (FM) algorithm for local search. This basic configuration is then successively extended with specific components. The results of our experiments are summarized in Table 2 for increasing scaling parameter α'. In configuration Constant128 all components are enabled (+F,+M,+FM) and flow-based refinements are performed every 128 uncontractions. It is used as a reference point for the quality achievable using flow-based refinement.

The results indicate that only using flow-based refinement (+F,-M,-FM) 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 α'), the average connectivity is still worse than the reference configuration. Enabling the most balanced minimum cut heuristic improves partitioning quality. Configuration (+F,+M,-FM) performs better than the basic configuration for $\alpha' \geq 8$. By combining flows with the FM algorithm (+F,-M,+FM) we get a configuration that improves upon the baseline even for small flow problems. However, comparing this variant with (+F,+M,-FM) 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,+FM) 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,+FM) 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,+FM) with $\alpha' = 16$ for KaHyPar-MF. This configuration also performed best in the effectiveness tests presented in Appendix D. While this configuration performs better than KaHyPar-CA, its running time is still more than a factor of 3 higher.

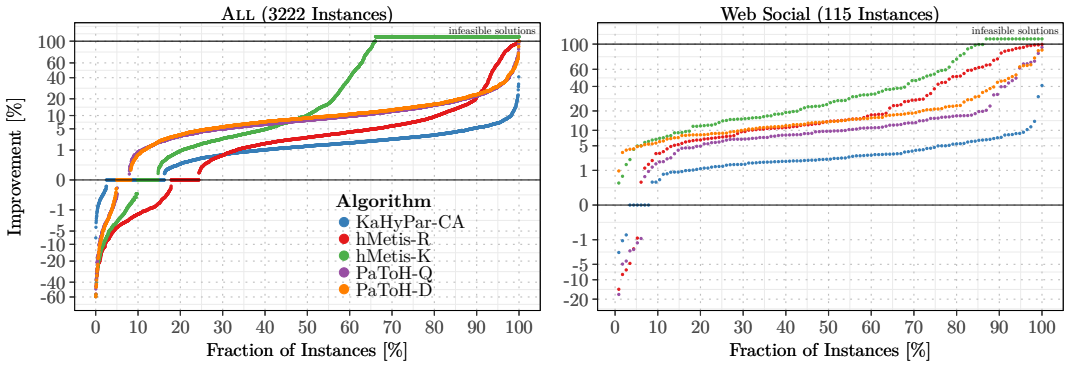

Figure 4 Improvement plots comparing KaHyPar-MF with KaHyPar-CA and other partitioners.

We therefore perform additional experiments on set B and successively enable the speedup techniques described in Section 3.3. The results are summarized in Table 3. 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 2 in total, while still computing solutions of comparable quality. Thus in the comparisons with other systems, all heuristics are enabled.

4.3 Comparison with other Systems

We now compare KaHyPar-MF to the state-of-the-art HGP systems on the full benchmark set. The following comparison is based on 3222 instances, since we exclude the same 194 out of 3416 instances that were already excluded in [25]. As can be seen in Figure 4 (left), KaHyPar-MF outperforms all other algorithms. Comparing the best solutions of KaHyPar-MF to each partitioner individually across all instances, it produced *better* partitions than PaToH-Q, PaToH-D, hMetis-K, KaHyPar-CA, hMetis-R for 92.1%, 91.7%, 85.3%, 83.8%, and 75.6% of the instances, respectively. Comparing the best solutions of all systems simultaneously, KaHyPar-MF produced the best partitions for 2427 of the 3222 instances. As can be seen in Figure 4 (right), the improvement is most pronounced for hypergraphs derived from matrices of web graphs and social networks[3], which are difficult to partition due to skewed degree and net size distributions. With 55.7s, the average running time of KaHyPar-MF is less than a factor of two slower than KaHyPar-CA, which took 31.1s. The average running times of hMetis-R, hMetis-K, PaToH-Q and PaToH-D were 79.2s, 57.9s, 5.9s and 1.2s, respectively. A detailed comparison of running times and solution quality can be found in Appendix E.

5 Conclusion

We generalize KaFFPa's flow-based refinement framework [44] 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 hypergraph flow networks.

[3] Based on the following matrices: `webbase-1M`, `ca-CondMat`, `soc-sign-epinions`, `wb-edu`, `IMDB`, `as-22july06`, `as-caida`, `astro-ph`, `HEP-th`, `Oregon-1`, `Reuters911`, `PGPgiantcompo`, `NotreDame_www`, `NotreDame_actors`, `p2p-Gnutella25`, `Stanford`, `cnr-2000`.

Furthermore we identify shortcomings of the KaFFPa [44] approach that restrict feasible solutions and introduce an advanced model that overcomes these limitations by utilizing the structure of hypergraph flow networks. Lastly, we present techniques to improve the running time of the framework by a factor of 2 without affecting solution quality. The resulting hypergraph partitioner KaHyPar-MF performs better than all competing algorithms and has a running time comparable to that of hMetis. Since our flow model yields better solutions for both hypergraphs *and* graphs than the KaFFPa approach, future work includes the integration of our flow model into KaFFPa and the evaluation in the context of graph partitioning. We also plan to extend our framework to optimize other objectives such as cut.

References

1. P. Agrawal, B. Narendran, and N. Shivakumar. Multi-way partitioning of VLSI circuits. In *9th International Conference on VLSI Design*, pages 393–399, 1996.
2. Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag. Engineering a direct k-way Hypergraph Partitioning Algorithm. In *19th Workshop on Algorithm Engineering and Experiments, (ALENEX)*, pages 28–42, 2017.
3. C. J. Alpert. The ISPD98 Circuit Benchmark Suite. In *International Symposium on Physical Design, (ISPD)*, pages 80–85, 1998.
4. C. J. Alpert, J.-H. Huang, and A. B. Kahng. Multilevel Circuit Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, 1998.
5. C. J. Alpert and A. B. Kahng. Recent Directions in Netlist Partitioning: a Survey. *Integration, the VLSI Journal*, 19(1–2):1–81, 1995.
6. C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level Direct K-way Hypergraph Partitioning with Multiple Constraints and Fixed Vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, 2008.
7. D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Proc. Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*. AMS, 2013. `doi:10.1090/conm/588`.
8. A. Belov, D. Diepold, M. Heule, and M. Järvisalo. The SAT Competition 2014. `http://www.satcompetition.org/2014/`, 2014.
9. T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452, 1993.
10. Ü. V. Catalyürek and C. Aykanat. Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, Jul 1999.
11. J. Cong and M. Smith. A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design. In *30th Conference on Design Automation, (DAC)*, pages 755–760, 1993.
12. N. J. Cox. Stata tip 96: Cube roots. *Stata Journal*, 11(1):149–154(6), 2011. URL: `http://www.stata-journal.com/article.html?article=st0223`.
13. T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011.
14. K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and Ü. V. Catalyürek. Parallel Hypergraph Partitioning for Scientific Computing. In *20th International Conference on Parallel and Distributed Processing, (IPDPS)*, pages 124–124. IEEE, 2006.
15. W.E. Donath. Logic partitioning. *Physical Design Automation of VLSI Systems*, pages 65–86, 1988.
16. J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19(2):248–264, 1972.

17 C. Fiduccia and R. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *19th ACM/IEEE Design Automation Conf.*, pages 175–181, 1982.

18 D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

19 Lester R Ford and Delbert R Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

20 A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

21 Andrew Goldberg, Sagi Hed, Haim Kaplan, Robert Tarjan, and Renato Werneck. Maximum Flows by Incremental Breadth-First Search. *19th European Symposium on Algorithms, (ESA)*, pages 457–468, 2011.

22 S. Hauck and G. Borriello. An Evaluation of Bipartitioning Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):849–866, 1997.

23 B. Hendrickson and R. Leland. A Multi-Level Algorithm For Partitioning Graphs. *1995 Conference on Supercomputing, (SC)*, 0:28, 1995.

24 T. Heuer. High Quality Hypergraph Partitioning via Max-Flow-Min-Cut Computations. Master's thesis, KIT, 2018.

25 T. Heuer and S. Schlag. Improving Coarsening Schemes for Hypergraph Partitioning by Exploiting Community Structure. In *16th International Symposium on Experimental Algorithms, (SEA)*, page 21:1–21:19, 2017.

26 Heuer, T. and Sanders, P. and Schlag, S. Network Flow-Based Refinement for Multilevel Hypergraph Partitioning, 2018. `arXiv:1802.03587`.

27 T. C. Hu and K. Moerder. Multiterminal Flows in a Hypergraph. In T.C. Hu and E.S. Kuh, editors, *VLSI Circuit Layout: Theory and Design*, chapter 3, pages 87–93. IEEE Press, 1985.

28 E. Ihler, D. Wagner, and F. Wagner. Modeling Hypergraphs by Graphs with the Same Mincut Properties. *Information Processing Letters*, 45(4):171–175, 1993. `doi:10.1016/0020-0190(93)90115-P`.

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

30 G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):69–79, 1999.

31 G. Karypis and V. Kumar. Multilevel K-way Hypergraph Partitioning. In *36th Design Automation Conference, (DAC)*, pages 343–348. ACM, 1999.

32 B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2):291–307, Feb 1970.

33 E. Lawler. Cutsets and Partitions of Hypergraphs. *Networks*, 3(3):275–285, 1973.

34 E. Lawler. *Combinatorial Optimization : Networks and Matroids*. Holt, Rinehart, and Whinston, 1976.

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

36 J. Li, J. Lillis, and C. K. Cheng. Linear decomposition algorithm for VLSI design applications. In *1995 International Conference on Computer Aided Design, (ICCAD)*, pages 223–228, Nov 1995.

37 H. Liu and D. F. Wong. Network-Flow-Based Multiway Partitioning with Area and Pin Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(1):50–59, Jan 1998.

38 Z. Mann and P. Papp. Formula partitioning revisited. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 41–56. EasyChair, 2014.

39 H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning Complex Networks via Size-Constrained Clustering. In *13th International Symposium on Experimental Algorithms, (SEA)*, pages 351–363, 2014.

40 D. A. Papa and I. L. Markov. Hypergraph Partitioning and Clustering. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007. doi:10.1201/9781420010749.

41 Jean-Claude Picard and Maurice Queyranne. On the Structure of all Minimum Cuts in a Network and Applications. *Combinatorial Optimization II*, pages 8–16, 1980.

42 Joachim Pistorius and Michel Minoux. An Improved Direct Labeling Method for the Max–Flow Min–Cut Computation in Large Hypergraphs and Applications. *International Transactions in Operational Research*, 10(1):1–11, 2003.

43 L. A. Sanchis. Multiple-way Network Partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

44 P. Sanders and C. Schulz. Engineering Multilevel Graph Partitioning Algorithms. In *19th European Symposium on Algorithms, (ESA)*, volume 6942 of *LNCS*, pages 469–480. Springer, 2011.

45 S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. k-way Hypergraph Partitioning via n-Level Recursive Bisection. In *18th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67, 2016.

46 D. G. Schweikert and B. W. Kernighan. A Proper Model for the Partitioning of Electrical Circuits. In *9th Design Automation Workshop, (DAC)*, pages 57–62. ACM, 1972.

47 A. Trifunovic. *Parallel Algorithms for Hypergraph Partitioning*. PhD thesis, University of London, 2006.

48 A. Trifunović and W. J. Knottenbelt. Parallel Multilevel Algorithms for Hypergraph Partitioning. *Journal of Parallel and Distributed Computing*, 68(5):563–581, 2008.

49 Ü. V. Çatalyürek and M. Deveci and K. Kaya and B. Uçar. UMPa: A multi-objective, multi-level partitioner for communication minimization. In Bader et al. [7], pages 53–66. doi:10.1090/conm/588.

50 B. Uçar and C. Aykanat. Encapsulating Multiple Communication-Cost Metrics in Partitioning Sparse Rectangular Matrices for Parallel Matrix-Vector Multiplies. *SIAM Journal on Scientific Computing*, 25(6):1837–1859, 2004.

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

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

53 S. Wichlund. On multilevel circuit partitioning. In *International Conference on Computer-Aided Design, (ICCAD)*, ICCAD, pages 505–511. ACM, 1998.

54 H. H. Yang and D. F. Wong. Efficient Network Flow Based Min-Cut Balanced Partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1533–1540, 1996.

A Framework Pseudocode

Algorithm 1: Flow-Based Refinement.

Input: Hypergraph H, k-way partition $\Pi_k = \{V_1, \ldots, V_k\}$, imbalance parameter ε.

Algorithm `MaxFlowMinCutRefinement`(H, Π_k)

```
Q := QuotientGraph(H, Π_k)
while ∃ active blocks ∈ Q do                    // in the beginning all blocks are active
    foreach {(V_i, V_j) ∈ Q | V_i ∨ V_j is active} do    // choose a pair of blocks
        Π_old = Π_best := {V_i, V_j} ⊆ Π_k       // extract bipartition to be improved
        ε_old = ε_best := imbalance(Π_k)        // imbalance of current k-way partition
        α := α'                                 // use large B-corridor for first iteration
        do                                      // adaptive flow iterations
            B := computeB-Corridor(H, Π_best, αε)   // as described in Section 2.3
            H_B := SubHypergraph(H, B)          // create B-induced subhypergraph
            N_B := FlowNetwork(H_B)             // as described in Section 3.1
            F := FlowProblem(N_B)               // as described in Section 3.2
            f := maxFlow(F)                     // compute maximum flow on F
            Π_f := mostBalancedMinCut(f, F)     // as in Section 2.3 & 3.1
            ε_f := imbalance(Π_f ∪ Π_k \ Π_old)  // imbalance of new k-way partition
            if (cut(Π_f) < cut(Π_best) ∧ ε_f ≤ ε) ∨ ε_f < ε_best then // found improvement
                α := min(2α, α'), Π_best := Π_f, ε_best := ε_f   // update α, Π_best, ε_best
            else α := α/2                       // decrease size of B-corridor in next iteration
        while α ≥ 1
        if Π_best ≠ Π_old then                  // improvement found
            Π_k := Π_best ∪ Π_k \ Π_old          // replace Π_old with Π_best
            activateForNextRound(V_i, V_j)      // reactivate blocks for next round
return Π_k
```

Output: improved ε-balanced k-way partition $\Pi_k = \{V_1, \ldots, V_k\}$

B Proof of Lemma 1

▶ **Lemma 1.** *Let f be a maximum (s,t)-flow in the Lawler network $\mathcal{N} = (\mathcal{V}, \mathcal{E})$ of a hypergraph $H = (V, E)$ and $(\mathcal{S}, \mathcal{V} \setminus \mathcal{S})$ be the corresponding (s,t)-min-cut in $\mathcal{N}$. Then for each node $v \in \mathcal{S} \cap V$, the residual graph $\mathcal{N}_f = (\mathcal{V}_f, \mathcal{E}_f)$ contains at least one path $\langle s, \ldots, e'' \rangle$ to a bridging node e'' of a net $e \in \mathrm{I}(v)$.*

Proof. Since $v \in \mathcal{S}$, there has to be some path $s \rightsquigarrow v$ in $\mathcal{N}_f$. By definition of the flow network, this path can either be of the form $P_1 = \langle s, \ldots, e'', v \rangle$ or $P_2 = \langle s, \ldots, e', v \rangle$ for some bridging nodes e', e'' corresponding to nets $e \in \mathrm{I}(v)$. In the former case we are done, since $e'' \in P_1$. In the latter case the existence of edge $(e', v) \in \mathcal{E}_f$ implies that there is a positive flow $f(v, e') > 0$ over edge $(v, e') \in \mathcal{E}$. Due to flow conservation, there exists at least one edge $(e'', v) \in \mathcal{E}$ with $f(e'', v) > 0$, which implies that $(v, e'') \in \mathcal{E}_f$. Thus we can extend the path P_2 to $\langle s, \ldots, e', v, e'' \rangle$. ◀

C Overview and Properties of Benchmark Sets

Sparse matrices (SPM) are translated into hypergraphs using the row-net model [10], 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 [40], while in the *primal* model each variable is represented by a vertex and each clause is represented by a net. In the *dual* model the opposite is the case [38].

Table 4 Overview about different benchmark sets. Set B and set C are subsets of set A.

	Source	#	DAC	ISPD98	Primal	Dual	Literal	SPM	Graphs
Set A	[25]	477	10	18	92	92	92	184	-
Set B	[25]	165	5	10	30	30	30	60	-
Set C	new	25	-	5	5	5	5	5	-
Set D	[39]	15	-	-	-	-	-	-	15

Table 5 Basic properties of set C. The number of pins is denoted with p.

Class	Hypergraph	n	m	p
ISPD	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
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
Literal	6s133	96 430	140 968	328 924
	6s153	171 292	245 440	572 692
	aaai10-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
Primal	6s153	85 646	245 440	572 692
	aaai10-planning-ipc5	53 919	308 235	690 466
	hwmcc10-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
	mixtank_new	29 957	29 957	1 995 041
	laminar_duct3D	67 173	67 173	3 833 077

Table 6 Basic properties of the graph instances.

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

Table 7 Results of the effectiveness test for different configurations of our flow-based refinement framework for increasing α'. The quality in column Avg[%] is relative to the baseline configuration.

Config. α'	(+F,-M,-FM) Avg[%]	(+F,+M,-FM) Avg[%]	(+F,-M,+FM) Avg[%]	(+F,+M,+FM) 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,+FM)	6377.15		

D 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 [44], are presented in Table 7. Combinations of flows and FM local search perform better than repeated executions of the baseline configuration, with (+F,+M,-FM) and $\alpha' = 16$ performing best.

E Comparison of Running Times and Solution Quality

Table 8 Comparing the average running times of KaHyPar-MF with KaHyPar-CA and other hypergraph partitioners for different benchmark sets (top) and different values of k (bottom).

Algorithm	Running Time $t[s]$							
	All	DAC	ISPD98	Primal	Literal	Dual	SPM	WebSoc
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40	110.15
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91	67.14
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79	89.69
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95	111.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42	4.71
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77	0.88
	$k = 2$	$k = 4$	$k = 8$	$k = 16$	$k = 32$	$k = 64$	$k = 128$	
KaHyPar-MF	19.75	32.89	47.52	60.38	78.51	100.34	119.15	
KaHyPar-CA	12.68	17.16	23.88	31.01	41.69	57.35	76.61	
hMetis-R	27.87	51.59	74.74	91.09	109.13	128.66	149.34	
hMetis-K	25.47	32.27	42.50	53.41	74.00	109.12	152.92	
PaToH-Q	1.93	3.61	5.44	7.01	8.40	10.06	11.44	
PaToH-D	0.43	0.77	1.12	1.42	1.71	2.02	2.29	

Table 9 Comparing the best solutions of KaHyPar-MF with the best results of KaHyPar-CA and other partitioners for different benchmark sets (top) and different values of k (bottom). All values correspond to the quality improvement of KaHyPar-MF relative to the respective partitioner (in %).

Algorithm	Min. $(\lambda - 1)$							
	All	DAC	ISPD98	Primal	Literal	Dual	SPM	WebSoc
KaHyPar-MF	7542.88	16 828.15	5511.40	15 236.13	15 197.60	2927.42	6010.05	7478.06
KaHyPar-CA	2.22 %	2.80 %	1.92 %	1.85 %	2.46 %	3.33 %	1.74 %	3.91 %
hMetis-R	14.40 %	4.75 %	2.76 %	3.88 %	4.17 %	31.20 %	16.37 %	41.92 %
hMetis-K	12.92 %	7.77 %	2.17 %	4.78 %	6.91 %	21.51 %	16.23 %	40.45 %
PaToH-Q	11.48 %	15.24 %	9.53 %	14.36 %	14.98 %	11.44 %	8.36 %	18.45 %
PaToH-D	12.06 %	15.57 %	10.90 %	12.47 %	15.17 %	13.64 %	9.45 %	23.04 %
	$k = 2$	$k = 4$	$k = 8$	$k = 16$	$k = 32$	$k = 64$	$k = 128$	
KaHyPar-MF	1005.76	2985.22	5805.19	9097.31	14 352.34	21 537.33	31 312.48	
KaHyPar-CA	1.71 %	2.16 %	2.51 %	2.51 %	2.45 %	2.16 %	2.05 %	
hMetis-R	22.25 %	17.62 %	15.63 %	14.29 %	11.94 %	9.80 %	8.01 %	
hMetis-K	21.82 %	13.66 %	12.76 %	13.49 %	10.62 %	9.18 %	7.81 %	
PaToH-Q	14.92 %	12.60 %	11.81 %	11.66 %	10.66 %	9.77 %	8.63 %	
PaToH-D	8.54 %	10.41 %	13.64 %	14.50 %	12.70 %	12.66 %	11.89 %	