Network Flow-Based Refinement for Multilevel Hypergraph Partitioning

SEA’18 · June 27, 2018
Tobias Heuer, Peter Sanders, Sebastian Schlag
Hypergraphs

- generalization of graphs
  ⇒ hyperedges connect \( \geq 2 \) nodes

- graphs ⇒ dyadic (2-ary) relationships
- hypergraphs ⇒ (d-ary) relationships

- hypergraph \( H = (V, E, c, \omega) \)
  - vertex set \( V = \{1, \ldots, n\} \)
  - edge set \( E \subseteq \mathcal{P}(V) \setminus \emptyset \)
  - node weights \( c : V \to \mathbb{R}_{\geq 1} \)
  - edge weights \( \omega : E \to \mathbb{R}_{\geq 1} \)
Hypergraphs

- generalization of graphs
  ⇒ hyperedges connect ≥ 2 nodes

- graphs ⇒ dyadic (2-ary) relationships
- hypergraphs ⇒ (d-ary) relationships

- hypergraph $H = (V, E, c, \omega)$
  - vertex set $V = \{1, \ldots, n\}$
  - edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - node weights $c : V \to \mathbb{R}_{\geq 1}$
  - edge weights $\omega : E \to \mathbb{R}_{\geq 1}$
\( \varepsilon \)-Balanced Hypergraph Partitioning Problem

Partition hypergraph \( H = (V, E, c, \omega) \) into \( k \) disjoint blocks \( \Pi = \{ V_1, \ldots, V_k \} \) such that:

- blocks \( V_i \) are **roughly equal-sized**:

\[
c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil
\]

- connectivity objective is minimized.
\(\varepsilon\)-Balanced Hypergraph Partitioning Problem

Partition hypergraph \(H = (V, E, c, \omega)\) into \(k\) disjoint blocks \(\Pi = \{V_1, \ldots, V_k\}\) such that:

- blocks \(V_i\) are **roughly equal-sized**:
  \[
  c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil
  \]

**imbalance parameter**
ε-Balanced Hypergraph Partitioning Problem

Partition hypergraph $H = (V, E, c, \omega)$ into $k$ disjoint blocks $\Pi = \{ V_1, \ldots, V_k \}$ such that:

- blocks $V_i$ are **roughly equal-sized**:
  \[
  c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil
  \]

- **connectivity objective** is minimized:

\[\text{connectivity objective is minimized:}\]
**ε-Balanced Hypergraph Partitioning Problem**

Partition hypergraph $H = (V, E, c, \omega)$ into $k$ disjoint blocks $\Pi = \{ V_1, \ldots, V_k \}$ such that:

- blocks $V_i$ are **roughly equal-sized**:
  \[ c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil \]

- connectivity objective is **minimized**:
  \[ \sum_{e \in \text{cut}} (\lambda - 1) \omega(e) \]

connectivity: # blocks connected by net $e$
ε-Balanced Hypergraph Partitioning Problem

Partition hypergraph $H = (V, E, c, \omega)$ into $k$ disjoint blocks $\Pi = \{ V_1, \ldots, V_k \}$ such that:

- blocks $V_i$ are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

- **connectivity** objective is minimized:

$$\sum_{e \in \text{cut}} (\lambda - 1) \omega(e) = 7$$

# blocks connected by net $e$
Applications

- VLSI Design
- Warehouse Optimization
- Complex Networks
- Route Planning
- Simulation
- Scientific Computing

\[ \mathbb{R}^{n \times n} \ni Ax = b \in \mathbb{R}^n \]
The Multilevel Framework

input hypergraph

match /

contract

cluster

local search

uncontract

output partition

· · ·

initial partitioning

initial partitioning

· · ·
This Talk: Refinement Phase

input hypergraph

match /

cluster

contract

local search

uncontract

initial partitioning

output partition
State-of-the-Art Multilevel HGP Refinement

Algorithm 1: FM Local Search

while improvement found do
    while ¬ done do
        find best move
        perform best move
        rollback to best solution
    pass

pass

rollback
State-of-the-Art Multilevel HGP Refinement

Algorithm 1: FM Local Search

while improvement found do
    while ¬ done do
        find best move
        perform best move
        rollback to best solution
    pass

connectivity

pass 1    pass 2

vertex moves

rollback
State-of-the-Art Multilevel HGP Refinement

Algorithm 1: FM Local Search

while improvement found do
  while ¬ done do
    find best move
    perform best move
  rollback to best solution

Known Limitations:

× prone to get stuck in local optima

× large nets ⇝ zero gain moves
State-of-the-Art Multilevel HGP Refinement

Algorithm 1: FM Local Search

while improvement found do
  while ¬ done do
    find best move
    perform best move
  rollback to best solution

pass 1

connectivity

pass 2

vertex moves

Are there viable alternatives?

- prone to get stuck in local optima
- large nets $\sim$ zero gain moves
Flow-Based Refinement for Graph Partitioning

Goal: balanced partition with minimum cut makes the problem hard!
Flow-Based Refinement for Graph Partitioning

Goal: balanced partition with minimum cut

network flows
+ max-flow min-cut theorem
⇒ min. (s, t)-cuts

https://brilliant.org/wiki/max-flow-min-cut-algorithm/
Flow-Based Refinement for Graph Partitioning

Goal: balanced partition with minimum cut

network flows
+ max-flow min-cut theorem
⇒ min. \((s, t)\)-cuts

⇒ employed for graph partitioning in KaFFPa [Sanders, Schulz 11]

\(k\)-way refinement via \text{pairwise} flow-based improvements
The KaFFPa Framework [Sanders, Schulz 11]

select two adjacent blocks for refinement

build flow network

solve flow problem

find most-balanced minimum cut
Our Refinement Framework/ Contributions

As in KaFFPa, but
- speedup heuristics

select two adjacent blocks for refinement

Hypergraph Flow Networks
- size reduction

build flow network

Flow Problem
- improved model
- further size reduction

solve flow problem

find most-balanced minimum cut

As in KaFFPa
I am going to talk about...

As in KaFFPa, but

- speedup heuristics

select two adjacent blocks for refinement

Hypergraph Flow Networks

- size reduction

build flow network

Flow Problem

- improved model
- further size reduction

solve flow problem

find most-balanced minimum cut

As in KaFFPa
Hypergraph Flow Networks

Hypergraph $H$
Hypergraph Flow Networks: Star-Expansion $G^*$
Hypergraph Flow Networks: Lawler Network [Lawler 73]

⇒ node capacities \( \sim \) edge capacities
Hypergraph Flow Networks: Lawler Network [Lawler 73]

⇒ node capacities ~ edge capacities
Hypergraph Flow Networks: Lawler Network [Lawler 73]

⇒ node capacities $\sim$ edge capacities

$\omega(e) = \infty$
Hypergraph Flow Networks: Lawler Network [Lawler 73]

⇒ node capacities \(\sim\) edge capacities

⇒ hypernode \(v\) induces \(2d(v)\) edges
Hypergraph Flow Networks: Lawler Network \([\text{Lawler 73}]\)

\[ \Rightarrow \text{node capacities } \sim \Rightarrow \text{edge capacities} \]

\[ \Rightarrow \text{hypernode } v \text{ induces } 2d(v) \text{ edges} \]

\[ \Rightarrow \text{net } e \text{ induces } 2 \text{ nodes } \& 1 \text{ edge} \]
Hypergraph Flow Networks: Lawler Network [Lawler 73]

⇒ node capacities ↦ edge capacities

\[ S \rightarrow 1 \rightarrow t \]
Hypergraph Flow Networks: Liu-Wong Network \([LW98]\)

special treatment of **two-pin** nets

⇒ save 2 nodes + 3 edges
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]

Observation: min. \((s, t)\)-vertex separator has to be subset of star-nodes
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]
⇒ replace ∞-nodes with cliques...

\[ S \rightarrow t \]
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]
⇒ replace $\infty$-nodes with cliques...

... and apply Lawler transformation

... and apply Lawler transformation
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]
⇒ replace ∞-nodes with cliques...

⇒ removed hypernode v induces \(d(v)(d(v) - 1)\) edges
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]
⇒ replace ∞-nodes with cliques...

⇒ removed hypernode \( v \) induces \( d(v)(d(v) - 1) \) edges

If \( d(v) \leq 3 \), then \( d(v)(d(v) - 1) \leq 2d(v) \)
Hypergraph Flow Networks: Our Network

Minimum-Weight Vertex Separator [Hu, Moerder 85]
⇒ replace ∞-nodes with cliques...

⇒ removed hypernode \( v \) induces \( d(v)(d(v) - 1) \) edges

If \( d(v) \leq 3 \),
then \( d(v)(d(v) - 1) \leq 2d(v) \)

⇒ remove hypernodes with \( d(v) \leq 3 \)
Hypergraph Flow Networks: Our Network

⇒ combine low degree hypernode removal with Liu-Wong transformation
I am going to talk about ...

As in KaFFPa, but
- speedup heuristics

select two adjacent blocks for refinement

Hypergraph Flow Networks
- size reduction

build flow network

Flow Problem
- improved model
- further size reduction

solve flow problem

find most-balanced minimum cut
KaFFPa’s Flow-Based Refinement for hypergraphs

\[ V_i \quad \text{Cut} \quad B_1 \quad B_2 \quad V_j \]
KaFFPa’s Flow-Based Refinement for hypergraphs

construct area $B = B_1 \cup B_2$ s.t. every $(s,t)$-cut is $\varepsilon$-balanced in $H$

\[
c(B_1) \leq (1 + \varepsilon)\left\lceil \frac{c(V)}{k} \right\rceil - c(V_j) \leq
\]
KaFFPa’s Flow-Based Refinement for hypergraphs

construct area $B = B_1 \cup B_2$ s.t. every $(s,t)$-cut is $\epsilon$-balanced in $H$

$$c(B_1) \leq (1 + \epsilon) \left\lceil \frac{c(V)}{k} \right\rceil - c(V_j) \iff \Rightarrow c(B_2) \leq (1 + \epsilon) \left\lceil \frac{c(V)}{k} \right\rceil - c(V_i)$$
KaFFPa’s Flow-Based Refinement for hypergraphs

construct area \( B = B_1 \cup B_2 \) s.t. every \((s,t)\)-cut is \( \varepsilon \)-balanced in \( H \)

\[
c(B_1) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil - c(V_j) \iff c(B_2) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil - c(V_i)
\]
KaFFPa’s Flow-Based Refinement \textcolor{red}{\textbf{for hypergraphs}}

construct area $B = B_1 \cup B_2$ s.t. every (s,t)-cut is $\epsilon$-balanced in $H$

$$c(B_1) \leq (1 + \epsilon)\left\lceil \frac{c(V)}{k} \right\rceil - c(V_j) \iff \Rightarrow c(B_2) \leq (1 + \epsilon)\left\lceil \frac{c(V)}{k} \right\rceil - c(V_i)$$
KaFFPa’s Flow-Based Refinement for hypergraphs

build and solve flow problem

$V_i$ $V_j$

$B_1$ $B_2$

Cut
KaFFPa’s Flow-Based Refinement for hypergraphs

build and solve flow problem
KaFFPa’s Flow-Based Refinement for hypergraphs

build and solve flow problem

⇒ **optimal cut** in subhypergraph $\leadsto$ **improved** $\varepsilon$-balanced cut in $H$
Shortcomings of the KaFFPa Approach
Shortcomings of the KaFFPa Approach

(border nodes cannot move!)

⇒ no min-cuts involving border nets possible!
Shortcomings of the KaFFPa Approach

-border nodes cannot move!

⇒ no min-cuts involving border nets possible!
Shortcomings of the KaFFPa Approach

- Border nodes cannot move!
- No min-cuts involving border nets possible!

Lawler Network
Shortcomings of the KaFFPa Approach

- border nodes cannot move!

⇒ no min-cuts involving border nets possible!

particularly bad for

- large nets \(\leadsto\) likely to be border nets
- small imbalance \(\varepsilon\) \(\leadsto\) small area \(B\)
Shortcomings of the KaFFPa Approach

† border nodes cannot move!

⇒ no min-cuts involving border nets possible!

particularly bad for
- large nets \(\leadsto\) likely to be border nets
- small imbalance \(\epsilon\) \(\leadsto\) small area \(B\)

also applies to GP
Solution: A more flexible Model
Solution: A more flexible Model

extend flow problem to include border nets

Diagram showing a network flow problem with nodes Vi and Vj, with a cut between B1 and B2, and border nets extended to t.
Solution: A more flexible Model

extend flow problem to include border nets

⇒ border nodes become movable ✓
Solution: A more flexible Model

extend flow problem to include border nets

⇒ border nodes become movable ✓
⇒ but flow problem becomes larger ↘
Solution: A more flexible Model

remove border pins with help of $e'$, $e''$ nodes
Solution: A more flexible Model

remove border pins with help of $e'$, $e''$ nodes

$V_i$ $s$

$B_1$ $B_2$

Cut

Lawler Network

$\omega(e)$ $e'$ $e''$
Solution: A more flexible Model

remove border pins with help of $e'$, $e''$ nodes

Lawler Network
Solution: A more flexible Model

remove border pins with help of $e', e''$ nodes

Lawler Network
Solution: A more flexible Model

remove border pins with help of $e'$, $e''$ nodes

special case: single-pin border nets
Solution: A more flexible Model

remove border pins with help of $e', e''$ nodes

special case: single-pin border nets
A more flexible Model – Summary

✓ movable border nodes $\leadsto$ all cuts are feasible
✓ no increase in problem size
✓ further size reduction through $|e| = 1$ border nets
Experiments – Benchmark Setup

- System: 1 core of 2 Intel Xeon E5-2670 @ 2.6 Ghz, 64 GB RAM

- # (Hyper)graphs: [publicly available]
  - SuiteSparse Matrix Collection 184
  - SAT Competition 2014 (3 representations) 92
  - ISPD98 & DAC2012 VLSI Circuits 28
  - DIMACS Graphs [flow model experiments] 15

- \( k \in \{2, 4, 8, 16, 32, 64, 128\} \) with imbalance: \( \varepsilon = 3\% \)

- Comparing **KaHyPar-MF** with:
  - KaHyPar-CA
  - hMetis-R & hMetis-K
  - PaToH-Default & PaToH-Quality
Size Reduction Of Hypergraph Flow Networks

Max-Flow Computation, area $|B| = 25,000$ hypernodes

- $\mathcal{N}_L$: Lawler Network
- $\mathcal{N}_W$: Liu-Wong Network
- $\mathcal{N}_{\text{Our}}$: Our Network
- $\mathcal{N}_{\text{Our}}^1$: Our Network with $|e| = 1$ opt.
Size Reduction Of Hypergraph Flow Networks

Max-Flow Computation, area $|B| = 25.000$ hypernodes

$N_L$: Lawler Network
$N_W$: Liu-Wong Network
$N_{\text{Our}}$: Our Network
$N^1_{\text{Our}}$: Our Network with $|e| = 1$ opt.

$\text{DAC}$
$\text{ISPD}$
$\text{Dual}$
$\text{Primal}$
$\text{Literal}$
$\text{SPM}$

$d(v)$: ↑
$|e|$: ↑

$d(v)$: ↓
$|e|$: ↓

$d(v)$: ↓
$|e|$: ↑
Impact of KaHyPar’s Flow Model

Average Improvement [%] over the KaFFPα Approach

<table>
<thead>
<tr>
<th>$\alpha'$</th>
<th>$\varepsilon = 1%$</th>
<th>$\varepsilon = 3%$</th>
<th>$\varepsilon = 5%$</th>
<th>$\varepsilon = 1%$</th>
<th>$\varepsilon = 3%$</th>
<th>$\varepsilon = 5%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.7</td>
<td>8.1</td>
<td>7.6</td>
<td>11.7</td>
<td>11.3</td>
<td>10.5</td>
</tr>
<tr>
<td>2</td>
<td>7.9</td>
<td>6.6</td>
<td>4.8</td>
<td>11.0</td>
<td>9.1</td>
<td>7.8</td>
</tr>
<tr>
<td>4</td>
<td>6.9</td>
<td>3.9</td>
<td>2.7</td>
<td>9.9</td>
<td>7.3</td>
<td>5.4</td>
</tr>
<tr>
<td>8</td>
<td>5.1</td>
<td>2.3</td>
<td>1.5</td>
<td>8.6</td>
<td>5.3</td>
<td>3.9</td>
</tr>
<tr>
<td>16</td>
<td>3.4</td>
<td>1.3</td>
<td>1.2</td>
<td>7.0</td>
<td>4.1</td>
<td>3.5</td>
</tr>
</tbody>
</table>

⇒ performs **better** on all problem sizes and imbalances
⇒ most pronounced for **small** flow problems & imbalances
⇒ effects also visible for **graphs**
Impact of KaHyPar’s Flow Model

Average Improvement [%] over the KaFFPpa Approach

<table>
<thead>
<tr>
<th>α'</th>
<th>Hypergraphs</th>
<th></th>
<th></th>
<th>DIMACS Graphs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.7</td>
<td>8.1</td>
<td>7.6</td>
<td>11.7</td>
<td>11.3</td>
<td>10.5</td>
</tr>
<tr>
<td>2</td>
<td>7.9</td>
<td>6.6</td>
<td>4.8</td>
<td>11.0</td>
<td>9.1</td>
<td>7.8</td>
</tr>
<tr>
<td>4</td>
<td>6.9</td>
<td>3.9</td>
<td>2.7</td>
<td>9.9</td>
<td>7.3</td>
<td>5.4</td>
</tr>
<tr>
<td>8</td>
<td>5.1</td>
<td>2.3</td>
<td>1.5</td>
<td>8.6</td>
<td>5.3</td>
<td>3.9</td>
</tr>
<tr>
<td>16</td>
<td>3.4</td>
<td>1.3</td>
<td>1.2</td>
<td>7.0</td>
<td>4.1</td>
<td>3.5</td>
</tr>
</tbody>
</table>

⇒ performs **better** on **all** problem sizes and imbalances
⇒ most pronounced for **small** flow problems & imbalances
⇒ effects also visible for **graphs**
Impact of KaHyPar’s Flow Model

Average Improvement [%] over the KaFFPa Approach

<table>
<thead>
<tr>
<th>α′</th>
<th>Hypergraphs</th>
<th>DIMACS Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ε = 1%</td>
<td>ε = 3%</td>
</tr>
<tr>
<td>1</td>
<td>7.7</td>
<td>8.1</td>
</tr>
<tr>
<td>2</td>
<td>7.9</td>
<td>6.6</td>
</tr>
<tr>
<td>4</td>
<td>6.9</td>
<td>3.9</td>
</tr>
<tr>
<td>8</td>
<td>5.1</td>
<td>2.3</td>
</tr>
<tr>
<td>16</td>
<td>3.4</td>
<td>1.3</td>
</tr>
</tbody>
</table>

⇒ performs **better** on all problem sizes and imbalances
⇒ most pronounced for **small** flow problems & imbalances
⇒ effects also visible for **graphs**
State-of-the-Art: HGP Quality

Algorithm
- KaHyPar-CA
- hMetis-R
- hMetis-K
- PaToH-Q
- PaToH-D

infeasible solutions

# Instances

1- (Best/Algorithm)
KaHyPar-MF: HGP Quality & Running Time

![Graph showing the quality and running time of different algorithms for 3222 instances.]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>t[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaHyPar-MF</td>
<td>55.67</td>
</tr>
<tr>
<td>KaHyPar-CA</td>
<td>31.05</td>
</tr>
<tr>
<td>hMetis-R</td>
<td>79.23</td>
</tr>
<tr>
<td>hMetis-K</td>
<td>57.86</td>
</tr>
<tr>
<td>PaToH-Q</td>
<td>5.89</td>
</tr>
<tr>
<td>PaToH-D</td>
<td>1.22</td>
</tr>
</tbody>
</table>

Infeasible solutions

Algorithm performance across 3222 instances.
Conclusion & Discussion

KaHyPar-MF – direct $k$-way HGP with **flow-based** refinement

- generalizes KaFFPa’s flow refinement to hypergraphs
- sparsified hypergraph flow networks
- improved flow model

In the paper / technical report:

- speedup heuristics $\sim$ factor 2 faster
- min-cut reconstruction
- more experimental results:
  - size of flow networks
  - different algorithm configurations
  - quality & running times per instance class

KaHyPar-Framework
Open-Source:
http://kahypar.org
Implementation Details

output partition

local search

uncontract

...
Implementation Details

KaFFPa
multi-level

log(n) flow + FM refinements

uncontract

local search

output partition

Flow
FM

Flow
FM

Flow
FM
Implementation Details

KaFFPa multi-level

KaHyPar n-level

output partition

local search

uncontract

... uncontractions

flow refinement after $2^i$ uncontractions
Implementation Details

KaFFPa
multi-level

KaHyPar
n-level

FM refinements
inbetween

output partition

local search

uncontract