Advanced Flow-Based Multilevel Hypergraph Partitioning

SEA 2020
June 5, 2020
Lars Gottesbüren, Michael Hamann, Sebastian Schlag, Dorothea Wagner
Hypergraph Partitioning

- Hypergraph \( H = (V, E, c, \omega) \)
- vertex set \( V = \{1, \ldots, n\} \)
- edge set \( E \subseteq \mathcal{P}(V) \setminus \emptyset \)
- incident edges \( \Gamma(u) = \{ e \in E \mid u \in e \} \)
- vertex weights \( \varphi : V \rightarrow \mathbb{R}_{\geq 1} \)
- edge weights \( \omega : E \rightarrow \mathbb{R}_{\geq 1} \)
Hypergraph Partitioning

- Hypergraph $H = (V, E, c, \omega)$
  - vertex set $V = \{1, \ldots, n\}$
  - edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - incident edges $\Gamma(u) = \{e \in E \mid u \in e\}$
  - vertex weights $\varphi : V \to \mathbb{R}_{\geq 1}$
  - edge weights $\omega : E \to \mathbb{R}_{\geq 1}$

![Diagram of hypergraph partitioning with pins and hyperedges]
Hypergraph Partitioning

Partition into $k$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$

- blocks $V_i$ have roughly equal weight:

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

Partition into $k$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$

- blocks $V_i$ have **roughly equal weight**:

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

imbalance
Hypergraph Partitioning

Partition into $k$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$

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

- **minimize connectivity** objective:
  \[
  \text{con} = \sum_{e \in E} (\lambda(e) - 1) \omega(e)
  \]
Hypergraph Partitioning

Partition into $k$ disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$

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

- minimize **connectivity** objective:
  \[
  \text{con} = \sum_{e \in E} (\lambda(e) - 1) \omega(e)
  \]

\[
\lambda(e) = \left| \{V_i \mid V_i \cap e \neq \emptyset\} \right|
\]

# blocks overlapping with $e$
Applications

Distributed Databases

Route Planning

VLSI Design

HPC
Multilevel Algorithms

Input Hypergraph

match or cluster

contract

local search

uncontract

initial partition

3
Multilevel Algorithms

Input Hypergraph

match or cluster

contract

local search

uncontract

initial partition

initial partition

contract
Classic Fiduccia-Mattheyses

Algorithm 1: FM Local Search

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

vertex moves

connectivity

rollback

slide kindly provided by Sebastian Schlag
Classic Fiduccia-Mattheyses

Algorithm 1: FM Local Search

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

pass 1  pass 2
vertex moves
connectivity
Classic Fiduccia-Mattheyses

Algorithm 1: FM Local Search

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

× get stuck in local optima

× large edges \(\rightsimeq\) zero gain moves

slide kindly provided by Sebastian Schlag
Classic Fiduccia-Mattheyses

Algorithm 1: FM Local Search

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

× get stuck in local optima

× large edges $\sim \rightarrow$ zero gain moves

max-flow-min-cut to the rescue

slide kindly provided by Sebastian Schlag
Classic Fiduccia-Mattheyses

Algorithm 1: FM Local Search

1. while improvement found do
2. while ¬ done do
3. find best move
4. perform best move
5. rollback to best solution

max-flow-min-cut to the rescue

Issues?
- only 2-way
- what are flows on hypergraphs?
- not balanced

get stuck in local optima?

large edges ⇒ zero gain moves

pass

vertex moves

X

connectivity

pass

pass

pass 1

pass 2

vertex moves

max-flow-min-cut to the rescue

slide kindly provided by Sebastian Schlag
Flow-Based Refinement in KaHyPar

- Select two adjacent blocks for refinement.
- Build graph-based flow model.
- Solve flow problem.
- Find more balanced minimum cut.

Slide kindly provided by Sebastian Schlag.
Flow-Based Refinement in KaHyPar

- either: restrict flow model size so that balance is guaranteed
- or: make it a little larger, hope for balance. if not ⇝ scale down again

(build graph-based flow model)

Slide kindly provided by Sebastian Schlag
Flow-Based Refinement in KaHyPar

select two adjacent blocks for refinement

build graph-based flow model

solve flow problem

find more balanced minimum cut

slide kindly provided by Sebastian Schlag
Flow-Based Refinement in KaHyPar

- Select two adjacent blocks for refinement.
- Build graph-based flow model.
- Solve flow problem.
- Find more balanced minimum cut.

Slide kindly provided by Sebastian Schlag
The new KaHyPar-HFC

select two adjacent blocks for refinement

flows directly on hypergraph

use FlowCutter

[Hamann, Strasser JEA18]

[Gottesbüren, Hamann, Wagner ESA19]

naturally built-in

find more balanced minimum cut
The new KaHyPar-HFC

select two adjacent blocks for refinement

flows directly on hypergraph

use FlowCutter
[Hamann, Strasser JEA18]
[Gottesbüren, Hamann, Wagner ESA19]
The new KaHyPar-HFC

select two adjacent blocks for refinement

flows directly on hypergraph

use FlowCutter
[Hamann, Strasser JEA18]
[Gottesbüren, Hamann, Wagner ESA19]

what’s new for FlowCutter?
- weighted instances
- new guidance
Flows on Hypergraphs

\[
\begin{array}{c}
\text{u} \\
\text{v} \\
\text{e} \\
\text{w} \\
\text{x}
\end{array}
\]
Flows on Hypergraphs

- $\tilde{f}(u, e) > 0 \Rightarrow u$ sends flow into $e$
- $\tilde{f}(u, e) < 0 \Rightarrow u$ receives flow from $e$
Flows on Hypergraphs

- $\tilde{f}(u, e) > 0 \Rightarrow u$ sends flow into $e$
- $\tilde{f}(u, e) < 0 \Rightarrow u$ receives flow from $e$
- $\text{rcap}(e, u, v) = c(e) - f(e) + \tilde{f}(u, e)^- + \tilde{f}(v, e)^+ = (12 - 7) + 2 + 4 = 11$
- $\max(0, -\tilde{f}(u, e)) \quad \max(0, \tilde{f}(v, e))$
Flows on Hypergraphs

- $\tilde{f}(u, e) > 0 \Rightarrow u$ sends flow into $e$
- $\tilde{f}(u, e) < 0 \Rightarrow u$ receives flow from $e$

$rcap(e, u, v) = c(e) - f(e) + \tilde{f}(u, e)^- + \tilde{f}(v, e)^+ = (12 - 7) + 2 + 4 = 11$

$max(0, -\tilde{f}(u, e))$  $max(0, \tilde{f}(v, e))$
Flows on Hypergraphs

- $\tilde{f}(u, e) > 0 \Rightarrow u$ sends flow into $e$
- $\tilde{f}(u, e) < 0 \Rightarrow u$ receives flow from $e$

$$\text{rcap}(e, u, v) = c(e) - f(e) + \tilde{f}(u, e) - + \tilde{f}(v, e) + = (12 - 7) + 2 + 4 = 11$$

$$\max(0, -\tilde{f}(u, e)) \quad \max(0, \tilde{f}(v, e))$$

$$f(e) = 7 \quad c(e) = 12$$

$\tilde{f}(u, e) = -2$  $\tilde{f}(x, e) = 3$  $\tilde{f}(v, e) = 4$  $\tilde{f}(w, e) = -5$  $f(e) = 7$  $c(e) = 12$
Flows on Hypergraphs

- \( \tilde{f}(u, e) > 0 \) \( \Rightarrow \) \( u \) sends flow into \( e \)
- \( \tilde{f}(u, e) < 0 \) \( \Rightarrow \) \( u \) receives flow from \( e \)
- \( rcap(e, u, v) = c(e) - f(e) + \tilde{f}(u, e)^- + \tilde{f}(v, e)^+ = (12 - 7) + 2 + 4 = 11 \)
- \( \max(0, -\tilde{f}(u, e)) \) \( \max(0, \tilde{f}(v, e)) \)
Flows on Hypergraphs

- \( \tilde{f}(u, e) > 0 \Rightarrow u \) sends flow into \( e \)
- \( \tilde{f}(u, e) < 0 \Rightarrow u \) receives flow from \( e \)
- \( rcap(e, u, v) = c(e) - f(e) + \tilde{f}(u, e)^- + \tilde{f}(v, e)^+ = (12 - 7) + 2 + 4 = 11 \)
- \( \max(0, -\tilde{f}(u, e)) \) \( \max(0, \tilde{f}(v, e)) \)
- can implement any flow algorithm by treating nets like vertices
- we use Dinic
Dinic

- residual BFS to compute distance labels
- residual DFS to find edge-disjoint shortest augmenting paths
- repeat until no flow augmented
Dinic

- residual BFS to compute distance labels
- residual DFS to find edge-disjoint shortest augmenting paths
- repeat until no flow augmented

Distance labels for hypergraphs

- $d[u]$ for vertices
- $d_i[e]$ for pushing flow to flow-sending pins $\tilde{f}(v, e) > 0$
- $d_o[e]$ for pushing flow to all pins. set if $c(e) - f(e) + \tilde{f}(u, e)^- > 0$
Dinic

- residual BFS to compute distance labels
- residual DFS to find edge-disjoint shortest augmenting paths
- repeat until no flow augmented

Optimizations

- capacity scaling
  - require $\geq \alpha$ residual capacity
  - if no flow augmented try with $\alpha \leftarrow \alpha/2$
- iterative DFS with stored iterators
- no allocations
- range of active values trick $\rightsquigarrow$ fast resets
FlowCutter
FlowCutter

1. Augment flow
FlowCutter

1. Augment flow

2. Find min s- and t-cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow  
2. Find min s- and t-cut  
3. Pick smaller side  
4. Assimilate side  
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut

creates augmenting path ⇒ bad candidate
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut

- avoids augmenting paths ⇒ good candidate
- creates augmenting path ⇒ bad candidate
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min $s$- and $t$-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
FlowCutter

1. Augment flow
2. Find min s- and t-cut
3. Pick smaller side
4. Assimilate side
5. Pierce cut
Experimental Setup

- Intel Xeon E5-2670 @ 2.6 Ghz, 64 GB RAM, 20 MB L3, 256KB L2
- # Hypergraphs: [publicly available]
  - SuiteSparse Matrix Collection 184
  - SAT Competition 2014 (3 representations) 92
  - ISPD98 & DAC2012 VLSI Circuits 28
- $k \in \{2, 4, 8, 16, 32, 64, 128\}$ with imbalance: $\varepsilon = 3\%$
- 10 random seeds
Experimental Setup

- Intel Xeon E5-2670 @ 2.6 Ghz, 64 GB RAM, 20 MB L3, 256KB L2
- # Hypergraphs: [publicly available]
  - SuiteSparse Matrix Collection 184
  - SAT Competition 2014 (3 representations) 92
  - ISPD98 & DAC2012 VLSI Circuits 28
- $k \in \{2, 4, 8, 16, 32, 64, 128\}$ with imbalance: $\varepsilon = 3\%$
- 10 random seeds
- Comparing **KaHyPar-HFC* and KaHyPar-HFC** with:
  - KaHyPar-MF
  - hMetis-R(ecursive Bisection) & hMetis-K (direct -kway)
  - PaToH-D(efault) & PaToH-Q(uality)
  - Mondriaan
  - Zoltan-AlgD
  - HYPE
Comparison with previous KaHyPar
Comparison with previous KaHyPar

Performance ratio $r(a, i) = \frac{\text{con}(a, i)}{\min\{\text{con}(a', i) \mid a' \in A\}}$ of algorithm $a$ on instance $i$
Comparison with previous KaHyPar

- Performance ratio $r(a, i) = \frac{\text{con}(a, i)}{\min\{\text{con}(a', i) | a' \in A\}}$ of algorithm $a$ on instance $i$

- Y-axis = fraction of instances with smaller performance ratio than value on x-axis
Comparison with previous KaHyPar

- performance ratio $r(a, i) = \frac{\text{con}(a,i)}{\min\{\text{con}(a',i) \mid a' \in A\}}$ of algorithm $a$ on instance $i$

- y-axis = fraction of instances with smaller performance ratio than value on x-axis

- $x = 1 \implies$ fraction of instances on which algorithm is the best

- higher is better
## Comparison with previous KaHyPar

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(t) [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaHyPar-MF</td>
<td>67.07</td>
</tr>
<tr>
<td>KaHyPar-HFC*</td>
<td>62.49</td>
</tr>
<tr>
<td>KaHyPar-HFC</td>
<td>44.84</td>
</tr>
</tbody>
</table>

![Graph comparing performance ratios of different algorithms](image-url)
Comparison with previous KaHyPar

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$t$ [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaHyPar-MF</td>
<td>67.07</td>
</tr>
<tr>
<td>KaHyPar-HFC*</td>
<td>62.49</td>
</tr>
<tr>
<td>KaHyPar-HFC</td>
<td>44.84</td>
</tr>
</tbody>
</table>

take with a grain of salt!
Comparison with other Partitioners

Performance ratio

Fraction of instances

- KaHyPar-HFC*
- KaHyPar-MF
- hMetis-K
- Zoltan-AlgD
- PaToH-D
- KaHyPar-HFC
- hMetis-R
- PaToH-Q
- Mondriaan
- HYPE
Comparison with other Partitioners

![Graph comparing partitioning performance](chart.png)

- **KaHyPar-HFC**
- **KaHyPar-MF**
- **hMetis-K**
- **Zoltan-AlgD**
- **PaToH-D**
- **PaToH-Q**
- **Mondriaan**
- **HYPE**

Legend:
- *infeasible*
- *timeout*
## Comparison with other Partitioners

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$t[s]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaHyPar-MF</td>
<td>67.07</td>
</tr>
<tr>
<td>KaHyPar-HFC*</td>
<td>62.49</td>
</tr>
<tr>
<td>KaHyPar-HFC</td>
<td>44.84</td>
</tr>
<tr>
<td>hMetis-R</td>
<td>96.55</td>
</tr>
<tr>
<td>hMetis-K</td>
<td>73.65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$t[s]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoltan-AlgD</td>
<td>107.60</td>
</tr>
<tr>
<td>PaToH-Q</td>
<td>7.48</td>
</tr>
<tr>
<td>PaToH-D</td>
<td>1.47</td>
</tr>
<tr>
<td>Mondriaan</td>
<td>6.44</td>
</tr>
<tr>
<td>HYPE</td>
<td>1.03</td>
</tr>
</tbody>
</table>
Detailed Running Time

![Box plot of running times for various algorithms](image-url)

- KaHyPar-MF
- KaHyPar-HFC
- hMetis-R
- hMetis-K
- Zoltan-AlgD
- PaToH-Q
- PaToH-D
- Mondriaan
- HYPE

Time [s]
Conclusion

KaHyPar-HFC – better and faster
Conclusion

**KaHyPar-HFC** – better and faster

In the TR:
- configuration study / assess impact of components
- different k / instance classes ⇒ improves a lot on dual SAT and large k
- earlier balance with subset sum for special vertices
- distance-based piercing
- flow routing
Conclusion

KaHyPar-HFC – better and faster

In the TR
- configuration study / assess impact of components
- different k / instance classes ⇒ improves a lot on dual SAT and large k
- earlier balance with subset sum for special vertices
- distance-based piercing
- flow routing

https://kahypar.org

https://github.com/kahypar/kahypar

https://github.com/larsgottesbueren/WHFC
Conclusion

KaHyPar-HFC – better and faster

In the TR
- configuration study / assess impact of components
- different k / instance classes ⇒ improves a lot on dual SAT and large k
- earlier balance with subset sum for special vertices
- distance-based piercing
- flow routing

- parallel versions coming soon™

https://kahypar.org
https://github.com/kahypar/kahypar
https://github.com/larsgottesbueren/WHFC