High Quality Graph Partitioning

Peter Sanders, Christian Schulz
ISMP 2012
Mesh partitioned
1. nodes ↔ data, computation
2. edges ↔ interdependencies

All PE’s get same amount of work
Communication is expensive

Graph Partitioning Problem:
Partition a graph into (almost) equally sized blocks, such that the number of edges connecting vertices from different blocks is minimal.
\( \epsilon \)-Balanced Graph Partitioning

Partition graph \( G = (V, E, c : V \rightarrow \mathbb{R}_{>0}, \omega : E \rightarrow \mathbb{R}_{>0}) \) into \( k \) disjoint blocks s.t.:

- total node weight of each block \( \leq \frac{1 + \epsilon}{k} \) total node weight
- total weight of cut edges as small as possible

Applications:
linear equation systems, VLSI design, route planning, …
Overview

- Introduction
- Multilevel Algorithms
- Advanced Techniques
- Evolutionary Techniques
- Experiments
- Summary
Multi-Level Graph Partitioning

Successful in existing algorithms: Metis, Scotch, DiBaP, . . ., KaPPa, KaSPar, KaFFPa, KaFFPaE
**Advanced Techniques**

**Talk Today**

- GP Folklore
- Edge Ratings
- Flow Based Refinements
- More Localized Local Search
- F-cycles for Graph Partitioning
But how are the edges selected?
Graph Partitioning
Matching Selection

Goals:
1. large edge weights $\leadsto$ sparsify
2. large #edges $\leadsto$ few levels
3. uniform node weights $\leadsto$ “represent” input
4. small node degrees $\leadsto$ “represent” input
$\leadsto$ unclear objective
$\leadsto$ gap to approx. weighted matching
which only considers 1., 2.

Our Solution:
Apply approx. weighted matching to general edge rating function
Graph Partitioning

Edge Ratings

\[
\omega(\{u, v\})
\]
\[
\text{expansion}^*(\{u, v\}) := \frac{\omega(\{u, v\})}{c(u)c(v)}
\]
\[
\text{expansion}^{*2}(\{u, v\}) := \frac{\omega(\{u, v\})^2}{c(u)c(v)}
\]
\[
\text{innerOuter}(\{u, v\}) := \frac{\omega(\{u, v\})}{\text{Out}(v) + \text{Out}(u) - 2\omega(u, v)}
\]

where \(c = \text{node weight}, \ \omega = \text{edge weight}, \ \text{Out}(u) := \sum_{\{u, v\} \in E} \omega(\{u, v\})\)
Initial Partitioning

Usually done by recursive bipartitioning, e.g. using BFS

- we currently use Scotch [Pellegrini]
- multiple tries pay off

Open Problem:
Direct $k$-partitioner that achieves better quality or speed.
compute $\text{gain} \ \forall \ \nu \in V$
FM Local Search

- compute gain $\forall v \in V$
- $g(v) = d_{ext}(v) - d_{int}(v)$
FM Local Search

- compute gain $\forall v \in V$
- $g(v) = d_{ext}(v) - d_{int}(v)$
- store gain of boundary nodes (e.g. in a heap)
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

Step: 0
Edge Cut: 5
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>Edge Cut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>
FM Local Search

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

<table>
<thead>
<tr>
<th>Step:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
FM Local Search

- stop after limit
- take best edge cut
- within balance constraint

<table>
<thead>
<tr>
<th>Step</th>
<th>Edge Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

edge cut

steps
FM Local Search

- stop after limit
- take best edge cut
- within balance constraint

<table>
<thead>
<tr>
<th>Step</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
FM Local Search – Discussion

+ Generalizable for multiple blocks
+ Linear time
– Unlikely to find improvements requiring \( \geq 2 \) negative gain moves
More Localized Local Search
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rand}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in_{\text{rnd}} T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in \text{rnd } T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in \text{rnd } T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in_{\text{rand}} T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
More Localized Local Search
inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in_{\text{rnd}} T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
More Localized Local Search
inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary ⇒ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rand}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

inspired by $n$-level search

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search
inspired by \( n \)-level search

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. complete boundary \( \Rightarrow \) maintained todo list \( T \)
  2. initialize search with single node \( v \in_{\text{rnd}} T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
Flows as Local Improvement
Flows as Local Improvement
Two Blocks

- area $B$, such that each $(s, t)$-min cut is $\epsilon$-balanced cut in $G$
- e.g. 2 times BFS (left, right)
Flows as Local Improvement
Two Blocks

- obtain optimal cut in $B$
- since each cut in $B$ yields a feasible partition → improved two-partition
- advanced techniques possible and necessary
- combination with local search works best
Example

100x100 Grid
Example

Constructed Flow Problem (using BFS)
Example
Apply Max-Flow Min-Cut
Example
Output Improved Partition
Local Improvement for $k$-partitions
Using Flows?

on each pair of blocks
Iterated Multilevel
Iterated Multilevel [Walshaw 2004]

- don’t contract cut edges
- adapt previous solution as initial partitioning
- cuts can only improve
- V-cycles / F-cycles
Evolutionary Techniques
Distributed Evolutionary Graph Partitioning

- **Evolutionary Algorithms:**
  - population of individuals
  - selection (based on fitness), mutation, recombination, ...

- **Evolutionary Graph Partitioning:**
  - individuals $\leftrightarrow$ partitions
  - fitness $\leftrightarrow$ edge cut
Combine

match

contract

- two individuals $P_1$, $P_2$:
  don’t contract cut edges of $P_1$ or $P_2$
- until no matchable edge is left
- coarsest graph $\leftrightarrow$ Q-graph of overlay
- $\rightarrow$ exchanging good parts is easy
- initial solution: use better of both parents
Example

Two Individuals $P_1, P_2$
Example

Overlay of $P_1$, $P_2$
Example
Multilevel Combine of $P_1, P_2$
Exchanging good parts is easy

Coarsest Level

- >> large weight, < small weight
- start with the better partition (red, $P_2$)
- move $v_4$ to the opposite block
- integrated into multilevel scheme (+local search on each level)
Example
Result of $\mathcal{P}_1, \mathcal{P}_2$
Parallelization

- each PE has its own island (a local population)
- locally: perform combine and mutation operations
- communicate analog to *randomized rumor spreading*
  1. rumor ↔ currently best local partition
  2. local best partition changed → send it to $\mathcal{O}(\log P)$ random PEs
  3. asynchronous communication (MPI Isend)
→ quality records in a few minutes for small graphs
Experiments
Example

Street network Europe $|V| = 18M, |E| = 44M, k = 64$
Buffoon $\leftrightarrow$ kMetis

edge cut 3825

depth cut 10264
Experimental Results
Comparison with Other Systems

Geometric mean, imbalance $\epsilon = 0.03$:
11 graphs (78K–18M nodes) $\times k \in \{2, 4, 8, 16, 32, 64\}$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>large graphs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Avg.</td>
<td>t[s]</td>
</tr>
<tr>
<td>KaFFPa strong</td>
<td>12 053</td>
<td>12 182</td>
<td>121.22</td>
</tr>
<tr>
<td>KaSPar strong</td>
<td>12 450</td>
<td>+3%</td>
<td>87.12</td>
</tr>
<tr>
<td>KaFFPa eco</td>
<td>12 763</td>
<td>+6%</td>
<td>3.82</td>
</tr>
<tr>
<td>Scotch</td>
<td>14 218</td>
<td>+20%</td>
<td>3.55</td>
</tr>
<tr>
<td>KaFFa fast</td>
<td>15 124</td>
<td>+24%</td>
<td>0.98</td>
</tr>
<tr>
<td>kMetis</td>
<td>15 167</td>
<td>+33%</td>
<td>0.83</td>
</tr>
</tbody>
</table>

- Walshaw instances, road networks, Florida Sparse Matrix Collection, random Delaunay triangulations, random geometric graphs
KaFFPaEvolutionary $k=64$

mean min cut

normalized time $t_n$

Repetitions

KaFFPaE
Walshaw Benchmark

- 816 instances ($\epsilon \in \{0, 1\%, 3\%, 5\%\}$)
- focus on partition quality

- Overall quality records $\leq$:

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$\leq$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>78%</td>
</tr>
<tr>
<td>1%</td>
<td>78%</td>
</tr>
<tr>
<td>3%</td>
<td>92%</td>
</tr>
<tr>
<td>5%</td>
<td>94%</td>
</tr>
</tbody>
</table>

new
Summary

input graph

flows etc. [ESA11]
local improvement

parallel [IPDPS10]
n-level [ESA10]

Cycles a la multigrid

Multilevel Graphpartitioning

[IPDPS10]
edge ratings
match +

contract

[SEA12]
initial partitioning
todo

Peter Sanders, Christian Schulz:
High Quality Graph Partitioning

Department of Informatics
Institute for Theoretical Computer Science, Algorithmics II
Current and Future Work

- $\epsilon = / \approx 0$
- open source release
- back to parallelization (+ external?)
- reconsider $n$-level? (flows?, . . . )
- other objective functions ((max.) communication volume, separators, . . . )
- hypergraph partitioning
- clustering
- mapping onto processors
- close gap to theory?
- etc.
Thank you!