Engineering
Graph Partitioning Algorithms

Peter Sanders, Christian Schulz
Motivating Example: Finite Element Simulation

- Simulation space is discretized into a **mesh**
- Solution of partial differential equations are approximated by linear equations
- Number of vertices can become quite large → **time and memory**
- Parallel processing required
Parallel FEM

- Mesh partitioned
  1. nodes ↔ data, computation
  2. edges ↔ interdependencies
- All PE’s get same amount of work
- Communication is expensive
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:
various numerical simulations, VLSI design, route planning, . . .
Multi-Level Graph Partitioning

Successful in existing systems:
Metis, Scotch, Jostle, . . . , KaPPa, KaSPar, KaFFPa, KaFFPaE
Overview

- Introduction
- Multilevel Algorithms
  - Coarsening
  - Initial Partitioning
  - Local Improvement
- Intelligently using multiple runs
- Parallelization
- Experiments
- Discussion
Coarsening – Goals

1. make/keep graph sparse
2. make/keep balanced node weights
3. few levels (?)
4. similar levels (?)

Multiple, partially conflicting criteria!
Common Approach – Contract Matchings

But how are the edges selected?
Edge Ratings

<table>
<thead>
<tr>
<th>Rating</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega({u, v})$</td>
<td>common but bad balance</td>
</tr>
<tr>
<td>$\omega({u, v})^2$</td>
<td>expansion $^2$</td>
</tr>
<tr>
<td>$\frac{c(u)c(v)}{2}$</td>
<td>for unit weights</td>
</tr>
<tr>
<td>$\omega({u, v})$</td>
<td>yesterday’s talk</td>
</tr>
<tr>
<td>$\frac{\text{Out}(v) + \text{Out}(u) - 2\omega(u, v)}{\text{Out}(v) + \text{Out}(u) - 2\omega(u, v)}$</td>
<td>algebraic distance</td>
</tr>
</tbody>
</table>

$c = \text{node weight}$

$\omega = \text{edge weight}$

$\text{Out}(u) := \sum_{\{u, v\} \in E} \omega(\{u, v\})$

**Open Problem:** Well understood unified rating function
Matching Selection

Two radically different Approaches:

Few Levels
Our interpretation:
Approximate max. overall rating matching
+ fast
+ parallelizable

\( n \) Levels
Just single most highly rated edge
[Osipov Sanders ESA 2010]
+ potentially high quality
+ conceptually simple
− slow? \( \rightsquigarrow \)
dynamic graph data structure, priority queue for edge selection
Node Centered Contraction

[Safro Sanders Schulz, SEA 12 (yesterday)]

- Select subset of **coarse nodes**
- Assign remaining nodes to one **or several** coarse nodes
- Multigrid inspired
  - Defers borderline decisions through fractional node assignment
  - Even fewer levels
    - too few ?
  - Currently slower than matching based coarsening
Overview

- Introduction
- Multilevel Algorithms
  - Coarsening
  - Initial Partitioning
  - Local Improvement
- Intelligently using multiple runs
- Parallelization
- Experiments
- Discussion
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.
Local Improvement – Overview

- FM local search [Fiduccia Mattheyses 1987]
- Localizing it for $n$-level partitioning
- Translating this to few levels
- Using flows
compute gain $\forall v \in V$
compute gain $\forall \ v \in V$

g(v) = d_{ext}(v) - d_{int}(v)$
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

<table>
<thead>
<tr>
<th>Step:</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</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>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Cut:</td>
<td>5</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>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>

![Graph Partitioning](image)
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>

![Graph diagram with edge cuts](image)

Graph representation with step progression and edge cut values.

Edge cut vs. steps graph showing the evolution of edge cuts over steps.
FM Local Search – Discussion

+ Generalizable for multiple blocks
+ Linear time
  – Unlikely to find improvements requiring $\geq 2$ negative gain moves
  – Too slow for $n$-level method
Focussing Local Search

Example: \( n \)-Level Search

Start at end points of uncontracted edge.

**Problem:** When to stop?

**Idea:** Get impatient when overall improvement seems unlikely. Estimate likelihood from a random walk with the same mean and variance as observed from the current walk.

\[ p\mu^2 \gg \sigma^2 ? \]
Back to Few Levels

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. boundary $\Rightarrow$ todo list $T$
  2. initialize search with random node from $T$
  3. iterate until $T = \emptyset$
- each node moved at most once
Back to Few Levels

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. boundary $\Rightarrow$ todo list $T$
  2. initialize search with random node from $T$
  3. iterate until $T = \emptyset$
- each node moved at most once
Back to Few Levels

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. \textit{boundary} \( \Rightarrow \) todo list \( T \)
  2. initialize search with random node from \( T \)
  3. iterate until \( T = \emptyset \)
- each node moved \textit{at most once}
Back to Few Levels

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. boundary \( \Rightarrow \) todo list \( T \)
  2. initialize search with random node from \( T \)
  3. iterate until \( T = \emptyset \)

- each node moved at most once
Back to Few Levels

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. boundary $\Rightarrow$ todo list $T$
  2. initialize search with random node from $T$
  3. iterate until $T = \emptyset$
- each node moved at most once
Back to Few Levels

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. boundary \( \Rightarrow \) todo list \( T \)
  2. initialize search with random node from \( T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
Back to Few Levels

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. boundary \( \Rightarrow \) todo list \( T \)
  2. initialize search with random node from \( T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
Back to Few Levels

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. boundary $\Rightarrow$ todo list $T$
  2. initialize search with random node from $T$
  3. iterate until $T = \emptyset$
- each node moved at most once
Back to Few Levels

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. \( \text{boundary} \Rightarrow \text{todo list } T \)
  2. initialize search with random node from \( T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
Back to Few Levels

- Typical: $k$-way local search initialized with complete boundary
- Localization:
  1. boundary $\Rightarrow$ todo list $T$
  2. initialize search with random node from $T$
  3. iterate until $T = \emptyset$
- each node moved at most once
Back to Few Levels

- Typical: \( k \)-way local search initialized with complete boundary
- Localization:
  1. boundary \( \Rightarrow \) todo list \( T \)
  2. initialize search with random node from \( T \)
  3. iterate until \( T = \emptyset \)
- each node moved at most once
Typical: $k$-way local search initialized with complete boundary

Localization:
1. boundary $\Rightarrow$ todo list $T$
2. initialize search with random node from $T$
3. iterate until $T = \emptyset$

each node moved at most once
Flows as Local Improvement
Two Blocks

- area $B$, such that every $(s, t)$-min cut is $\epsilon$-balanced cut in $G$
- e.g. 2 times BFS (left, right)
obtain optimal cut in $B$

since each cut in $B$ yields a feasible partition

→ improved two-partition

advanced techniques possible and necessary: most balanced minCut-heuristics, try larger $B$
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
Local Search or Flows?

<table>
<thead>
<tr>
<th>Local Search</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>global</td>
</tr>
<tr>
<td>multiway</td>
<td>two way</td>
</tr>
<tr>
<td>any $\epsilon$</td>
<td>large $\epsilon$</td>
</tr>
<tr>
<td>handicapped for $\epsilon \approx 0$</td>
<td></td>
</tr>
</tbody>
</table>

~~ Combination works best

**Open Problem/Current Work:** Really powerful technique for $\epsilon \approx 0$
Overview

- Introduction
- Multilevel Algorithms
  - Coarsening
  - Initial Partitioning
  - Local Improvement
- Intelligently using multiple runs
- Parallelization
- Experiments
- Discussion
Iterated Multilevel [Walshaw 2004]

- don’t contract cut edges
- adopt previous solution as initial partitioning
- cuts can only improve
- Analogy to Multigrid V-Cycles
Global Search
V-Cycles

Coarsening

Uncoarsening

○ Graph partitioned
• Graph not partitioned
Global Search

W-Cycles

Coarsening

Graph not partitioned

Uncoarsening

Graph partitioned

Graph partitioned

Graph not partitioned
Global Search

F-Cycles

Coarsening

Uncoarsening

Graph partitioned

Graph not partitioned
Distributed Evolutionary Graph Partitioning

- **Evolutionary Algorithms:**
  - highly inspired by biology
  - population of individuals
  - selection, mutation, recombination, ...
- **Goal:** Integrate KaFFPa into an Evolutionary Strategy
- **Evolutionary Graph Partitioning:**
  - individuals $\leftrightarrow$ partitions
  - fitness $\leftrightarrow$ edge cut
- **Parallelization** $\rightarrow$ quality records in a few minutes for small graphs
Combine

match

correct

down

- two individuals $\mathcal{P}_1$, $\mathcal{P}_2$: don’t contract cut edges of $\mathcal{P}_1$ or $\mathcal{P}_2$
- 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
Parallel Evolutionary Algorithm

- each PE has its own island (a local population)
- locally: perform combine and mutation operations
- communicate analogous 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)
Overview

- Introduction
- Multilevel Algorithms
  - Coarsening
  - Initial Partitioning
  - Local Improvement
- Intelligently using multiple runs
- Parallelization
- Experiments
- Discussion
Parallelization of Multilevel GP

Important case $k = p = \#\text{procs}$

[Holtgrewe Sanders Schulz IPDPS 2010]

Contraction: First locally then cross edges (assumes prepartitioning).

Current work: a simple natively parallel heuristics –
iterative local max

Initial Partitioning: $p$ replicated attempts

Local Improvement: Pairwise operations (FM search)

~ could be flows or focussed local searches
Pairwise Local Search

exchange boundaries

two local searches

adopt best

---

Sanders, Schulz:
Graph Partitioning
Finding Pairs of Blocks

- Color edges of quotient graph $Q$
  - several parallel algorithms tried
- Each color $\rightarrow$ matching in $Q$ $\rightarrow$ independent block pairs
Scalability

Street network Europe ($|V| = 18M, |E| = 44M$)
Overview

- Introduction
- Multilevel Algorithms
  - Coarsening
  - Initial Partitioning
  - Local Improvement
- Intelligently using multiple runs
- Parallelization
- Experiments
- Discussion
Example

Street network Europe $|V| = 18M$, $|E| = 44M$, $k = 64$
Buffoon ↔ kMetis
Experimental Results

Comparison with Other Systems

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

<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>KaFFPa 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
Walshaw Benchmark

Focus on partition quality.
34 Instances $\times k \in \{2, 4, 8, 16, 32, 64\} \times \epsilon \in \{0, 1\%, 3\%, 5\%\}$

<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
Larger more varied set of instances. F1 scoring system based on number of dominating other solvers. Metis and Scotch run by us.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Points</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaFFPaFast</td>
<td>1372</td>
<td>all solved</td>
</tr>
<tr>
<td>Metis</td>
<td>1265</td>
<td>some infeasible</td>
</tr>
<tr>
<td>KaFFPaEco</td>
<td>1174</td>
<td>missing largest instances\downarrow</td>
</tr>
<tr>
<td>KaFFPaE</td>
<td>1134</td>
<td>sometimes dominated by faster solver\downarrow</td>
</tr>
<tr>
<td>KaFFPaStrong</td>
<td>1085</td>
<td></td>
</tr>
<tr>
<td>UMPa</td>
<td>624</td>
<td></td>
</tr>
<tr>
<td>Scotch</td>
<td>361</td>
<td></td>
</tr>
<tr>
<td>Mondrian</td>
<td>225</td>
<td></td>
</tr>
</tbody>
</table>
Overview

- Introduction
- Multilevel Algorithms
  - Coarsening
  - Initial Partitioning
  - Local Improvement
- Intelligently using multiple runs
- Parallelization
- Experiments
- Discussion
Summary

input graph

flows etc. local improvement parallel

contract multigrid

initial partitioning todo

Multilevel Graphpartitioning

Cycles a la multigrid

output partition

edge ratings match +

[inIPDPS10] [inESA11] [inALENEX12]
Sources of Quality Improvement

- (better) edge ratings
- focussed local search with random walk stopping rule
- flows $\leftrightarrow n$-level
- better evolutionary operators (keep improving, no noise, . . . )
- parallelism
- better matchings
- $(F)$-cycles
Why Me?

- Many applications
- Established benchmarks
- Surprisingly diverse ingredients from the basic toolbox
  - priority queues
  - graph representation
  - BFS
  - weighted matchings
  - dominating sets
  - edge coloring
  - maximum flows
  - strongly connected components
  - shortest paths

local search
distributed, \( n \)-level
initial partitioning
coarsening
coarsening
parallelization
local improvement
balancing min-cuts
stay tuned
Current and Future Work

- $\epsilon = / \approx 0$
- initial partitioning
- open source release
- back to parallelization (+ external?)
- huge $k$
- reconsider $n$-level? (flows?, ...)
- other objective functions ((max.) communication volume, separators, ...)
- hypergraph partitioning
- clustering
- other multilevel applications (e.g., graph drawing)
- close gap to theory?
- etc.

WE ARE HIRING!
Quality
Evolutionary Graph Partitioning

<table>
<thead>
<tr>
<th>blocks $k$</th>
<th>KaFFPaE improvement over reps. of KaFFPa</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.2%</td>
</tr>
<tr>
<td>4</td>
<td>1.0%</td>
</tr>
<tr>
<td>8</td>
<td>1.5%</td>
</tr>
<tr>
<td>16</td>
<td>2.7%</td>
</tr>
<tr>
<td>32</td>
<td>3.4%</td>
</tr>
<tr>
<td>64</td>
<td>3.3%</td>
</tr>
<tr>
<td>128</td>
<td>3.9%</td>
</tr>
<tr>
<td>256</td>
<td>3.7%</td>
</tr>
<tr>
<td><strong>overall</strong></td>
<td><strong>2.5%</strong></td>
</tr>
</tbody>
</table>

2h time, 32 cores per graph and $k$, geom. mean
Quality
Of Parallel Partitioners

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

<table>
<thead>
<tr>
<th>Variant</th>
<th>avg. cut</th>
<th>balance</th>
<th>avg. time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaPPa-Strong</td>
<td>24 227</td>
<td>1.028</td>
<td>36.93</td>
</tr>
<tr>
<td>KaPPa-Fast</td>
<td>$+2%$</td>
<td>1.028</td>
<td>21.40</td>
</tr>
<tr>
<td>KaPPa-Minimal</td>
<td>$+3%$</td>
<td>1.028</td>
<td>5.94</td>
</tr>
<tr>
<td>seq. scotch</td>
<td>$+7%$</td>
<td>1.027</td>
<td>5.95</td>
</tr>
<tr>
<td>kmetis</td>
<td>$+18%$</td>
<td>1.026</td>
<td>0.79</td>
</tr>
<tr>
<td>parmetis</td>
<td>$+30%$</td>
<td>1.041</td>
<td>0.59</td>
</tr>
</tbody>
</table>

Walshaw instances, road networks, Florida Sparse Matrix Collection, random Delaunay triangulations, random geometric graphs, social networks.