High Quality Graph Partitioning

Peter Sanders, Christian Schulz
Overview

- Introduction
- Multilevel Algorithms
- Advanced Techniques
- Evolutionary Techniques
- Experiments
- Summary
\( \epsilon \)-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, …
Multi-Level Graph Partitioning

Successful in existing systems: Metis, Scotch, Jostle, ... , KaPPa, KaSPar, KaFFPa, KaFFPaE
Advanced Techniques
Talk Today

- Edge Ratings
- High Quality Matchings
- Flow Based Refinements
- More Localized Local Search
- F-cycles for Graph Partitioning
Graph Partitioning
Matching Selection

Goals:
1. large edge weights $\Rightarrow$ sparsify
2. large #edges $\Rightarrow$ few levels
3. uniform node weights $\Rightarrow$ “represent” input
4. small node degrees $\Rightarrow$ “represent” input

$\Rightarrow$ unclear objective
$\Rightarrow$ 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

\[ \omega(\{u, v\}) \]

\[ \text{expansion}(\{u, v\}) := \frac{\omega(\{u, v\})}{c(u) + c(v)} \]

\[ \text{expansion}^*(\{u, v\}) := \frac{\omega(\{u, v\})}{c(u)c(v)} \]

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

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

where \( c \) = node weight, \( \omega \) = edge weight,
\( \text{Out}(u) := \sum_{\{u,v\} \in E} \omega(\{u, v\}) \)
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)
- stop the BFS, if size would exceed $(1 + \epsilon) \frac{c(V)}{2} - c(V_2)$

$\Rightarrow c(V_{2_{\text{new}}}) \leq c(V_2) + (1 + \epsilon) \frac{c(V)}{2} - c(V_2)$
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
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

input graph

match

contract

initial partitioning

local improvement

output partition

...
More Localized Local Search

- **Idea:** *KaPPa, KaSPar* $\Rightarrow$ more local searches are better
- **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

- **Idea:** *KaPPa, KaSPar* $\Rightarrow$ more local searches are better
- **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

- **Idea:** *KaPPa, KaSPar* $\Rightarrow$ more local searches are better
- **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

- **Idea:** KaPPa, KaSPar $\Rightarrow$ more local searches are better
- **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

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

- Idea: KaPPa, KaSPar $\Rightarrow$ more local searches are better
- 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

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

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

- **Idea:** $KaPPa, KaSPar \Rightarrow$ more local searches are better
- **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

- **Idea:** KaPPa, KaSPar $\Rightarrow$ more local searches are better
- **Typical:** $k$-way local search initialized with complete boundary
- **Localization:**
  1. complete boundary $\Rightarrow$ maintained todo list $T$
  2. initialize search with single node $\nu \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$

- each node moved at most once
More Localized Local Search

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

- **Idea:** *KaPPa*, *KaSPar* ⇒ more local searches are better
- **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
Distributed Evolutionary Graph Partitioning

- Evolutionary Algorithms:
  - highly inspired by biology
  - population of individuals
  - selection, mutation, recombination, ...

- **Goal**: Integrate KaFFPa in 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

- 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

match

contract
Example

Two Individuals $\mathcal{P}_1, \mathcal{P}_2$
Example

Overlay of $\mathcal{P}_1$, $\mathcal{P}_2$
Example
Multilevel Combine of $P_1$, $P_2$
Exchanging good parts is easy
Coarsest Level

- $\gg \gg$ large weight, $<$ small weight
- start with the better partition ($\text{red, } P_2$)
- 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 $O(\log P)$ random PEs
  3. asynchronous communication (MPI Isend)
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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
</tr>
<tr>
<td>KaFFPa strong</td>
<td>12 053</td>
</tr>
<tr>
<td>KaSPar strong</td>
<td>12 450</td>
</tr>
<tr>
<td>KaFFPa eco</td>
<td>12 763</td>
</tr>
<tr>
<td>Scotch</td>
<td>14 218</td>
</tr>
<tr>
<td>KaFFa fast</td>
<td>15 124</td>
</tr>
<tr>
<td>kMetis</td>
<td>15 167</td>
</tr>
</tbody>
</table>

- Repeating Scotch as long as KaSPar strong run and choosing the best result $\sim 12.1\%$ larger cuts
- Walshaw instances, road networks, Florida Sparse Matrix Collection, random Delaunay triangulations, random geometric graphs
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>overall</td>
<td>2.5%</td>
</tr>
</tbody>
</table>

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

mean min cut

7900 8100 8300

normalized time $t_n$

Repetitions

KaFFPaE

$k=64$
Walshaw Benchmark

- runtime is not an issue
- 614 instances ($\epsilon \in \{1\%, 3\%, 5\%\}$)
- focus on partition quality

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$&lt;$</th>
<th>$\leq$</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaPPa</td>
<td>131</td>
<td>189</td>
</tr>
<tr>
<td>KaSPar</td>
<td>155</td>
<td>238</td>
</tr>
<tr>
<td>KaFFPa</td>
<td>317</td>
<td>435</td>
</tr>
<tr>
<td>KaFFPaE</td>
<td>300</td>
<td>470</td>
</tr>
</tbody>
</table>

- overall quality records $\leq$:

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$\leq$</th>
</tr>
</thead>
<tbody>
<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>
Summary

Distributed evolutionary algorithm [ALENEX12]

Cycles à la multigrid

Multilevel Graph Partitioning

Input graph

Match +

Contract

Local improvement

Uncontract

Initial partitioning

Output partition

Flows etc.
Outlook

- Further Material in the Paper(s)
  - F-cycles, High Quality Matchings, ....
  - Different combine and mutation operators
  - Specialization to road networks (Buffoon)
  - Many more details and experiments ...

- Future Work
  - other objective functions
    - currently via selection criterion
    - connectivity? \( \tilde{f}(\mathcal{P}) := f(\mathcal{P}) + \chi\{\mathcal{P} \text{ not connected}\} \cdot |E| \)
  - integrate other partitioners
  - graph clustering
  - open source release
Thank you!

Contact: christian.schulz@kit.edu
         sanders@kit.edu