Distributed Evolutionary Graph Partitioning

Peter Sanders, Christian Schulz
Workshop on EA & EA – Agra, India
Overview

- Introduction
- Multilevel Algorithms
- Evolutionary Techniques
- Experiments
- Summary
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.
The Common Parallel Approach

- Mesh partitioned via dual graph
  1. Each volume (data, calculation) is represented by a vertex
  2. Interdependencies are represented by edges
- 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.
\( \varepsilon \)-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 \( \leq \frac{1 + \varepsilon}{k} \) total node weight
- total weight of cut edges as small as possible

Applications:
linear equation systems, VLSI design, route planning, ...
Multilevel Graph Partitioning

1. Contraction
2. Local Search
3. Global Search
Multi-Level Graph Partitioning

Successful in existing systems: Metis, Scotch, Jostle, . . ., KaPPa, KaSPar, KaFFPa, KaFFPaE
Multilevel Graph Partitioning

1. Contraction
2. Local Search
3. Global Search
But how are the edges selected?
Graph Partitioning

Matching Selection

Goals:
1. large edge weights $\mapsto$ sparsify
2. large #edges $\mapsto$ few levels
3. uniform node weights $\mapsto$ “represent” input
4. small node degrees $\mapsto$ “represent” input
$\mapsto$ unclear objective
$\mapsto$ 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 \(c = \text{node weight}, \ \omega = \text{edge weight}, \ 
\text{Out}(u) := \sum_{\{u, v\} \in E} \omega(\{u, v\}) \)
Multilevel Graph Partitioning

1. Contraction
2. Local Search
3. Global Search
FM Local Search

\[ \text{compute gain } \forall v \in V \]
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>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>Edge Cut:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 2</td>
</tr>
<tr>
<td></td>
<td>5 4 6</td>
</tr>
</tbody>
</table>
FM Local Search

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

Step:
Edge Cut: 0 1 2 3
           5 4 6 8
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>

Graph with nodes connected by edges, and steps and edge cut values shown.
FM Local Search

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

Step: 0 1 2 3
Edge Cut: 5 4 6 8
FM Local Search – Discussion

+ Generalizable for multiple blocks
+ Linear time
- Unlikely to find improvements requiring $\geq 2$ negative gain moves
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 ⇒ 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 $v \in \text{rnd } T$
  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, KaSPa ⇒ 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 $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once
More Localized Local Search

- **Idea:** \textit{KaPPa, KaSPar} $\Rightarrow$ more local searches are better
- **Typical:** \(k\)-way local search initialized with \textit{complete boundary}
- **Localization:**
  1. \textit{complete boundary} $\Rightarrow$ maintained \textit{todo list} \(T\)
  2. initialize search with \textit{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
Multilevel Graph Partitioning

1. Contraction
2. Local Search
   - Localization of Local Search helps
   - Flows as Local Search
3. Global Search
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
  $\rightarrow$ 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
Multilevel Graph Partitioning

1. Contraction
2. Local Search
   - Localization of Local Search helps
   - Flows as Local Search
3. Global Search
Iterated Multilevel [Walshaw 2004]

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

- Parallelization → quality records in a few minutes for small graphs
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
Example

Two Individuals $\mathcal{P}_1, \mathcal{P}_2$
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 $\leftrightarrow$ currently best local partition
  2. local best partition *changed* $\rightarrow$ send it to $O(\log P)$ random PEs
  3. *asynchronous* communication (MPI Isend)
Experiments
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>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>

- 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>
</tbody>
</table>

| overall    | 2.5%                                     |

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

k=64

mean min cut

normalized time $t_n$

Repetitions

KaFFPaE
Scalability

mean min cut (mmc)

normalized time $t_n$

- $p = 1$
- $p = 2$
- $p = 4$
- $p = 8$
- $p = 16$
- $p = 32$
- $p = 64$
- $p = 128$
- $p = 256$
Walshaw Benchmark

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

- overall quality records (at submission time):

<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 Graph Partitioning

Department of Informatics
Institute for Theoretical Computer Science

input

ESA10

Output

Partition

contract

... ...

match
distr.
evol. Alg.

IPDPS10

Cycles a la multigrid

ESA11

n-level
todo

Parallel

Graphpartitioning
Multilevel

initial
partitioning

SEA12

edge
ratings

match+

[IPDPS10]

flows etc. [ESA11]

local improvement

[ESA11]

parallel [IPDPS10]

Contract

Output

ESA11

Graphpartitioning

Multilevel

Contract

Input

Graph
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