Overview

1. Introduction
   - Motivation
   - Balanced graph partitioning

2. Heuristics
   - Competitors
   - Stream ordering
   - Datasets

3. FENNEL

4. Conclusion
Motivation

Let’s take a look at Pregel:

- Pregel has to load the vertices
- and distribute them over its machines
- then graph processing can start
Motivation

Pregel uses a hash function $h : V \rightarrow \{1, \ldots k\}$

Now each machine has the same amount of vertices.

Now each machine has the same amount of vertices.
Motivation

- doesn’t minimize number of edges cut
- higher communication cost between machines afterwards
Motivation

- doesn’t minimize number of edges cut
- higher communication cost between machines afterwards

But: Pregel supports customized graph partitioning → use balanced graph partitioning
Let $k$ be the number of partitions we want to have. The goal is to minimize the cross partition edges and keeping the number of nodes in each partition even.

But: This is NP-hard, so in practise, only approximations are feasible.
Graph partitioning variants

We have two different approaches:

Offline

Online
Graph partitioning

- offline: know all vertices beforehand, then decide their places
- streaming: load the graph vertex after vertex and decide vertex place immediately

Offline algorithms are better at minimizing the amount of cut edges, streaming algorithm can be used during the same time pregel loads the graph into main memory.
METIS is widely used
combines various graph partition algorithms
needs access to all vertices at once, so it cannot be used during the loading phase of Pregel
We will compare the results of the streaming graph partitioning algorithms with those of METIS
Let $k$ be the number of partitions we want, $P = (P_1, \ldots, P_k)$ the partitions and $G = (V, E)$ the graph with $|V| = n, |E| = m$

Heuristics decide, in which partition $P_i$ a vertex $v \in V$ is put. We will see two kinds of heuristics: with a buffer and without one
Hashing

We use a fairly simple hash-function $H$: $V \rightarrow \{1, \ldots k\}$:

$$H(v) = (v \mod k) + 1$$

Hashing is mainly used in practise, because every machine knows where a node has been distributed to without the need of a mapping table.
Heuristics

Chunking

Let $C$ be the capacity of a Partition.
Then: divide stream into chunks of size $C$ and fill partitions in order

$$\text{index}(v) = (v \div C) + 1$$
Heuristics

**Deterministic Greedy**

Assign $v$ to the partition where it has the most edges.

Let $t$ be the time $v$ arrives, $N(v)$ be the set of vertices $v$ neighbours and $w$ a weight function penalizing partitions with too many vertices.

$$\text{index}(v) = \arg\max_{i \in [k]} \{|P_i \cap N(v)| \cdot w(i, t)|$$
Deterministic Greedy

Assign $v$ to the partition where it has the most edges. Let $t$ be the time $v$ arrives, $N(v)$ be the set of vertices $v$ neighbours and $w$ a weight function penalizing partitions with too many vertices.

$$\text{index}(v) = \arg\max_{i \in [k]} \{|P_i \cap N(v)| \times w(i, t)|$$

- $w(i, t) = 1$ for Unweighted Deterministic Greedy
- $w(i, t) = 1 - \frac{|P_i|}{C}$ for Linear Deterministic Greedy
- $w(i, t) = 1 - \exp\{|P_i| - C|$ for Exponentially Deterministic Greedy
Heuristics

This heuristic uses a Buffer of size C and assumes a way to differentiate between high-degree and low-degree nodes.

**Avoid Big**

We maintain the buffer and a threshold on large nodes. If the buffer is filled or the threshold is reached we greedily assign all low-degree nodes in the buffer. If only high-degree nodes remain, assign them with deterministic greedy.
Stream ordering means the order in which the vertices arrive at the streaming algorithm. Depending on the ordering, different heuristics achieve better or worse results → we have to specify the ordering

- Random: Standard Ordering in literature
- Breadth-First-Search ordering
- Depth-First-Search ordering
Datasets

- Marvel Comics social network, which resembles real social networks
  \[ |V| = 6,486, |E| = 427,018 \]
- PL, synthetic graphs generated by a power-law graph generator with clustering
  example P1000:
  \[ |V| = 1000, |E| = 9,878 \]
Evaluation

Marvel, $k = 8$
Evaluation

PL1000, $k = 4$
<table>
<thead>
<tr>
<th>Name</th>
<th>Random</th>
<th>BFS</th>
<th>DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid Big</td>
<td>-46.4</td>
<td>-27.3</td>
<td>-38.6</td>
</tr>
<tr>
<td>Chunking</td>
<td>0.7</td>
<td>37.6</td>
<td>35.7</td>
</tr>
<tr>
<td>Deterministic Greedy</td>
<td>45.4</td>
<td>57.7</td>
<td>54.7</td>
</tr>
<tr>
<td>Exp. Det. Greedy</td>
<td>47.5</td>
<td>59.4</td>
<td>56.2</td>
</tr>
<tr>
<td>Hashing</td>
<td>-1.7</td>
<td>-1.9</td>
<td>-2.1</td>
</tr>
<tr>
<td>Linear Det. Greedy</td>
<td>75.3</td>
<td>76</td>
<td>73</td>
</tr>
</tbody>
</table>
Results

- Linear Deterministic Greedy had the highest average gain, no matter what streaming ordering was used
- Nearly every heuristic is an improvement
- Except Avoid Big with negative improvement; this heuristic shouldn’t be used
Results

Why is Linear Deterministic Greedy better than the others?

**Deterministic Greed**

\[
\text{index}(v) = \arg \max_{i \in [k]} \{|P_i \cap N(v)| \times w(i, t)\}
\]

- \(w(i, t) = 1\) for Unweighted Det. Greedy
- \(w(i, t) = 1 - \frac{|P_i|}{C}\) for Linear Det. Greedy
- \(w(i, t) = 1 - \exp\{|P_i| - C\}\) for Exponentially Det. Greedy
Results

Why is Linear Deterministic Greedy better than the others?

- Unweighted Det. Greedy only indicates a already full partition
- Exponential Det. Greedy indicates a full partition only nearly at the limit
- Linear Det. Greedy prefers less loaded partitions
FENNEL combines two widely used families of heuristics:

- place newly arrived vertices in Partition with the largest number of neighbours
- place newly arrived vertices in Partition with the least non-neighbours
Let $P = (P_1, \ldots, P_k)$ be all $k > 0$ partitions, $P_i$ Partition $i$. Then we define a global objective function:

$$f(P) = c_{out}(P) + c_{in}(P)$$
\( c_{\text{out}}(P) = \#\text{edges cut} \)

is the inter-partition cost.

But what to do with \( c_{\text{in}}(P) \)?
But what to do with $c_{in}(P)$?

- Focus on individual partitions: $c_{in}(P) = \sum_{i=1}^{k} c_{in}(P_i)$
- Size of $P_i$ should be approximately $\frac{n}{k}$
We take a look at function family $c(x) = \alpha \cdot x^{\beta}$

- $\beta$ lets us control the importance of imbalance in the partition size
- $\beta = 1$ means ignoring imbalance of partition sizes
- $\beta = 2$ would diminish the importance of $c_{out}(P)$

Most of the time, the authors stuck to $\beta = 1.5$
We take a look at function family \( c(x) = \alpha \times x^\beta \)

\[
\alpha = m \times \frac{k^{\beta-1}}{n^\beta}, \text{ with } |E| = m
\]

with that we get:

\[
f(P) = c_{out}(P) + \sum_{i=1}^{k} c_{in}(P_i)
\]

\[
= \frac{\text{edges-cut}}{m} + \frac{1}{k} \sum_{i=1}^{k} \left( \frac{|P_i|}{n/k} \right)^\beta
\]
The graph partition problem with this objective function becomes:

$$\min_P \left( \frac{\text{#edges} - \text{cut}}{m} + \frac{1}{k} \sum_{i=1}^{k} \left( \frac{|P_i|}{n/k} \right)^\beta \right)$$

But this isn’t doable in a streaming algorithm!
Solution: We greedily decide, to which $P_i$ we assign an incoming vertex $v$

$$f((P_1, \ldots P_i \cup \{v\}, \ldots P_k)) \leq f((P_1, \ldots P_j \cup \{v\}, \ldots P_k), \ \forall j \in \{1, \ldots k\}$$

To stop overloading, we can add a threshold $t$:
If $|P_i| > t \cdot \frac{n}{k}$, $P_i$ can’t be chosen for $v$
Transform minimization problem into maximization:
\[ g((P_1, \ldots, P_i \cup \{v\}, \ldots, P_k)) \geq g((P_1, \ldots, P_j \cup \{v\}, \ldots, P_k)), \quad \forall j \in \{1, \ldots, k\} \]

This leads to:
\[ G(v, P_i) = |N(v) \cap P_i| - \alpha |P_i|^{\beta-1} \]
\[ G(v, P_i) = |N(v) \cap P_i| - \alpha \beta |P_i|^{\beta-1} \]

For \( \beta = 1 \) this is Deterministic Unweighted Greedy. For \( \beta = 2 = \frac{1}{\alpha} \) FENNEL would place vertices to the partitions with the least number of non-neighbours.
Now we compare FENNEL with Linear Det. Greedy and METIS ($k = 32$):

<table>
<thead>
<tr>
<th>Name</th>
<th>BFS % edges cut</th>
<th>BFS max partition load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Det. Greedy</td>
<td>34 %</td>
<td>1.01</td>
</tr>
<tr>
<td>FENNEL</td>
<td>14%</td>
<td>1.10</td>
</tr>
<tr>
<td>METIS</td>
<td>8%</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Now we compare FENNEL with Linear Det. Greedy and METIS ($k = 32$):

<table>
<thead>
<tr>
<th>Name</th>
<th>RANDOM % edges cut</th>
<th>RANDOM part. load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Det. Greedy</td>
<td>40 %</td>
<td>1.00</td>
</tr>
<tr>
<td>FENNEL</td>
<td>14%</td>
<td>1.02</td>
</tr>
<tr>
<td>METIS</td>
<td>8%</td>
<td>1.02</td>
</tr>
</tbody>
</table>
FENNEL

...and with METIS alone:

<table>
<thead>
<tr>
<th>$m$</th>
<th>$k$</th>
<th>$\lambda$ (Fennel)</th>
<th>$\rho$ (Fennel)</th>
<th>$\lambda$ (METIS)</th>
<th>$\rho$ (METIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7185314</td>
<td>4</td>
<td>62.5%</td>
<td>1.04</td>
<td>65.2%</td>
<td>1.02</td>
</tr>
<tr>
<td>6714510</td>
<td>8</td>
<td>82.2%</td>
<td>1.04</td>
<td>81.5%</td>
<td>1.02</td>
</tr>
<tr>
<td>6483201</td>
<td>16</td>
<td>92.9%</td>
<td>1.01</td>
<td>92.2%</td>
<td>1.02</td>
</tr>
<tr>
<td>6364819</td>
<td>32</td>
<td>96.3%</td>
<td>1.00</td>
<td>96.2%</td>
<td>1.02</td>
</tr>
<tr>
<td>6308013</td>
<td>64</td>
<td>98.2%</td>
<td>1.01</td>
<td>97.9%</td>
<td>1.02</td>
</tr>
<tr>
<td>6279566</td>
<td>128</td>
<td>98.4%</td>
<td>1.02</td>
<td>98.8%</td>
<td>1.02</td>
</tr>
</tbody>
</table>
Remember Pegel and others use a hash heuristic instead of FENNEL during the graph loading; now if we use FENNEL instead before a PageRank computation on LiveJournal:

<table>
<thead>
<tr>
<th># Clusters (k)</th>
<th>Run time [s]</th>
<th>Communication [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hash</td>
<td>FENNEL</td>
</tr>
<tr>
<td></td>
<td>Hash</td>
<td>FENNEL</td>
</tr>
<tr>
<td>4</td>
<td>32.27</td>
<td>25.49</td>
</tr>
<tr>
<td></td>
<td>321.41</td>
<td>196.9</td>
</tr>
<tr>
<td>8</td>
<td>17.26</td>
<td>15.14</td>
</tr>
<tr>
<td></td>
<td>285.35</td>
<td>180.02</td>
</tr>
<tr>
<td>16</td>
<td>10.64</td>
<td>9.05</td>
</tr>
<tr>
<td></td>
<td>222.28</td>
<td>148.67</td>
</tr>
</tbody>
</table>
Conclusion

- Despite a single pass, FENNEL can achieve results comparable to METIS, but is much faster.
- We have seen that using FENNEL actually improves graph operations like PageRank.
- This improvement stems entirely from the reduced network communication.