Streaming graph partitioning for large distributed graphs

Daniel Spanier

June 5, 2015

Overview

1. Introduction
   - Motivation
   - Balanced graph partitioning
2. Heuristics
   - Competitors
   - Stream ordering
   - Datasets
3. FENNEL
4. Conclusion

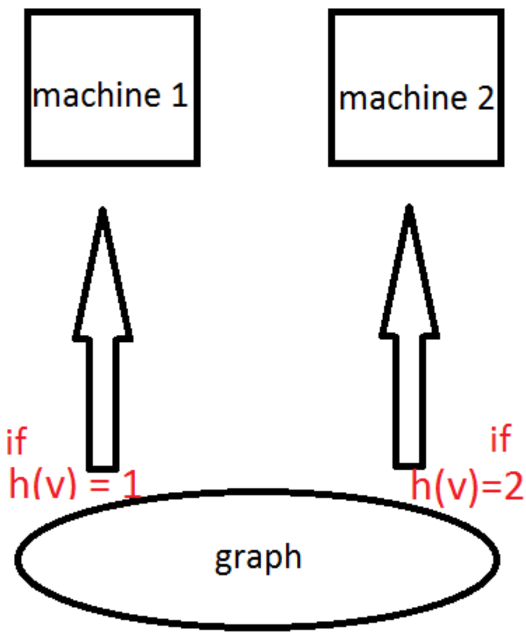
Motivation

Lets take a look at Pregel:

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

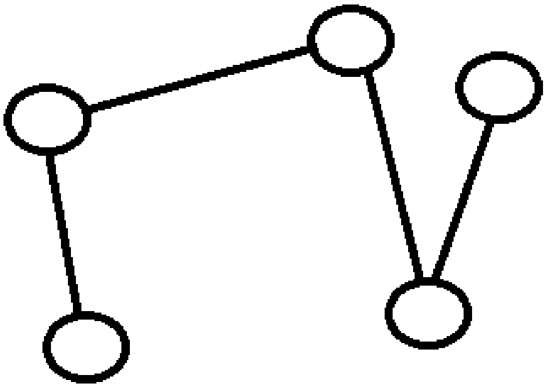
Motivation

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



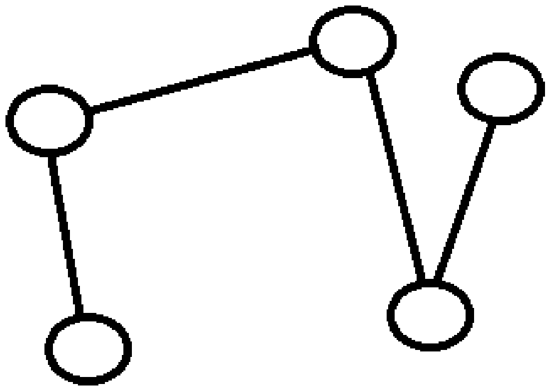
Now each machine has the same amount of vertices.

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

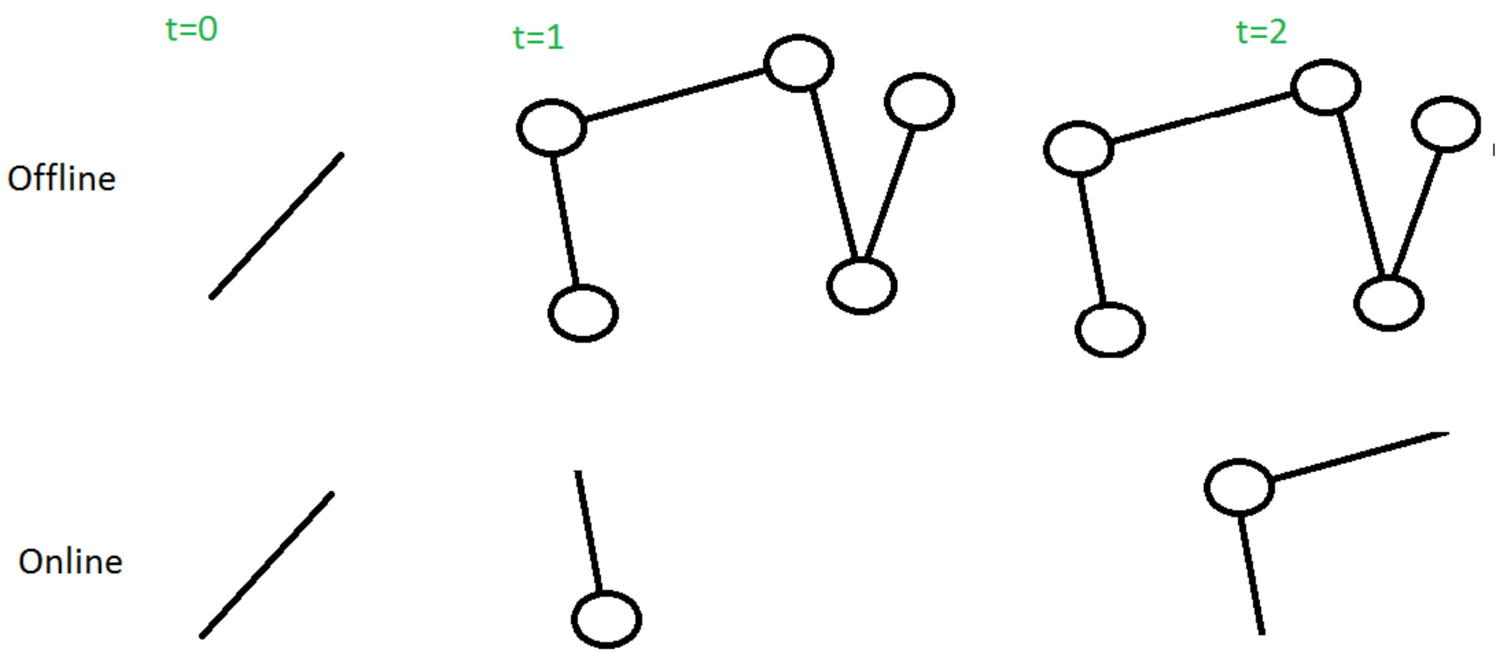
Balanced k-graph partitioning

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:



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.

Offline graph partitioning

- 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

Online graph partitioning

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

Heuristics

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
$index(v) = (v \ div \ C) + 1$

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.
$index(v) = \underset{i \in [k]}{argmax}\{|P_i \cap N(v)| * w(i,t)\}$

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.

$$index(v) = \underset{i \in [k]}{argmax}\{|P_i \cap N(v)| * w(i,t)\}$$

- $w(i,t) = 1$ 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 betweeen 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

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

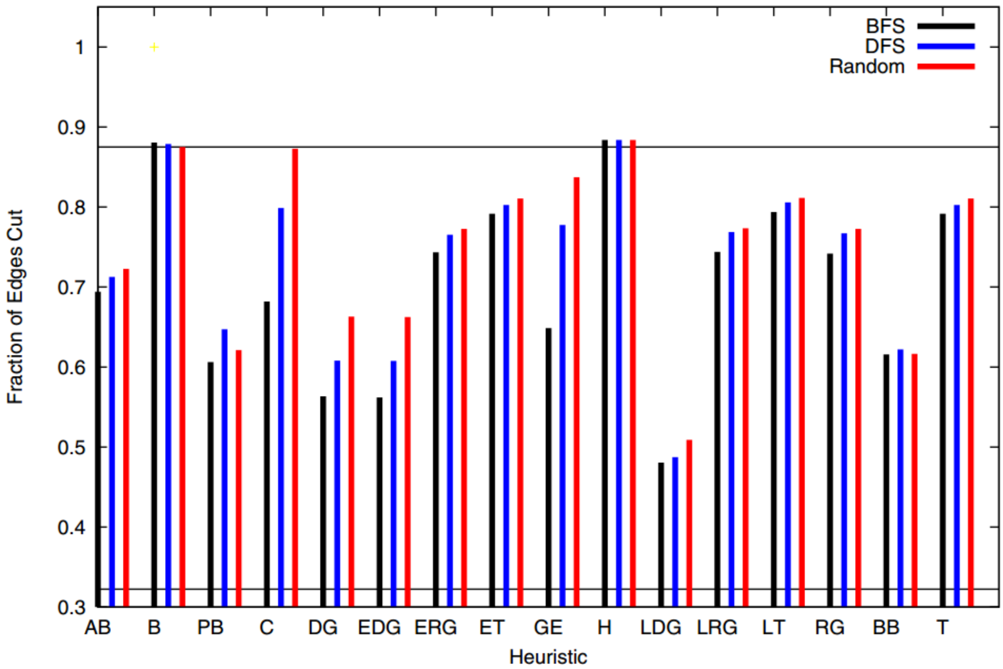
- Random: Standard Ordering in literature
- Breadth-First-Search ordering
- Depth-First-Search ordering

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:
  $|V| = 1000, |E| = 9,878$

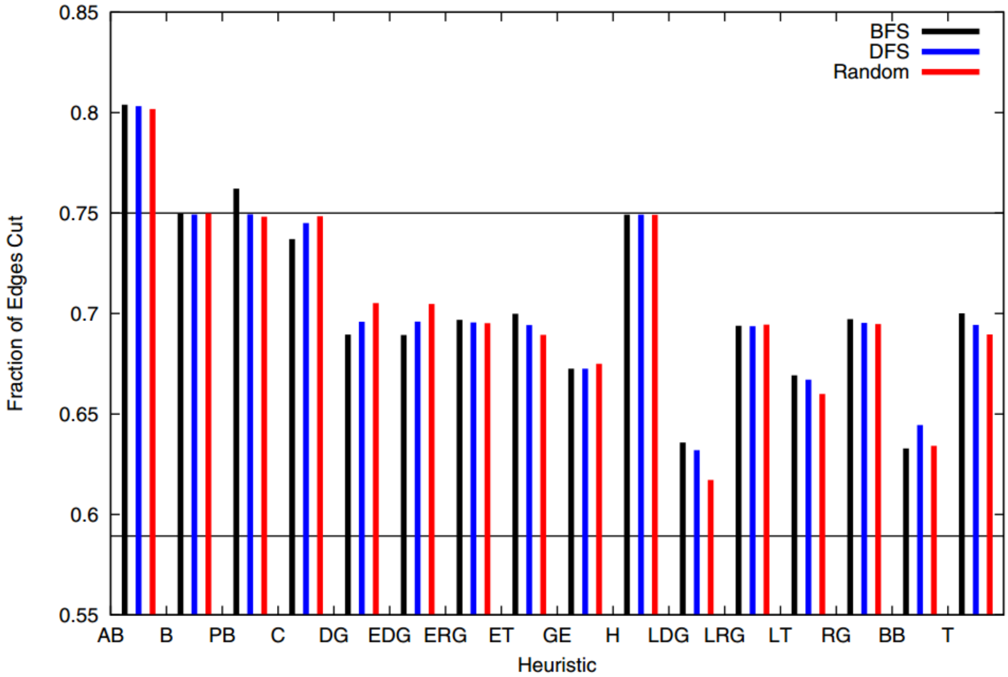
Evaluation

Marvel, $k = 8$



Evaluation

PL1000, $k = 4$



Evaluation

Name	**Random**	**BFS**	**DFS**
Avoid Big	-46.4	-27.3	-38.6
Chunking	0.7	37.6	35.7
Deterministic Greedy	45.4	57.7	54.7
Exp. Det. Greedy	47.5	59.4	56.2
Hashing	-1.7	-1.9	-2.1
Linear Det. Greedy	75.3	76	73

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

$$index(v) = \underset{i \in [k]}{argmax}\{|P_i \cap N(v)| * w(i,t)\}$$

- $w(i,t) = 1$ 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

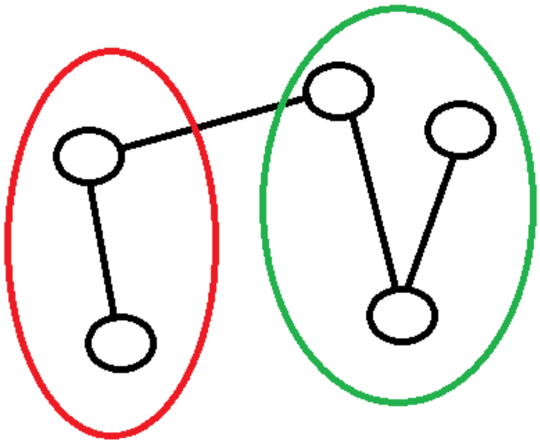
FENNEL combines two widely used families of heuristics:

- place newly arrived vertices in Partition with the largest number of neighbours
- place newly arrived vertices in Partition with the least non neighbours

FENNEL

Let $P = (P_1, ...P_k)$ be all $k > 0$ partitions, P_i Partition i.
Then we define a global objective function:

$$f(P) = c_{out}(P) + c_{in}(P)$$



FENNEL

$$c_{out}(P) = \#edges\ cut$$

is the inter-partition cost.
But what to do with $c_{in}(P)$?

FENNEL

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 $\frac{n}{k}$

FENNEL

We take a look at function family $c(x) = \alpha * x^{\beta}$

- β 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 $\beta = 1.5$

FENNEL

We take a look at function family $c(x) = \alpha * x^\beta$

$\alpha = m * \frac{k^{\beta-1}}{n^\beta}$, with $|E| = m$
with that we get:

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

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

FENNEL

The graph partition problem with this objective function becomes:

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

But this isn't doable in a streaming algorithm!

FENNEL

Solution: We greedily decide, to which P_i we assign an incoming vertex v

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

To stop overloading, we can add a threshold t:
If $|P_i| > t * \frac{n}{k}$, P_i can't be chosen for v

FENNEL

Transform minimization problem into maximization:
$g((P_1, \dots P_i \cup \{v\}, \dots P_k)) \geq g((P_1, \dots P_j \cup \{v\}, \dots P_k) \ , \quad \forall j \in \{1, \dots k\}$

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

FENNEL

$$G(v, P_i) = |N(v) \cap P_i| - \alpha\beta|P_i|^{\beta-1}$$

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

FENNEL

Now we compare FENNEL with Linear Det. Greedy and METIS ($k = 32$):

Name	**BFS % edges cut**	**BFS max partition load**
Linear Det. Greedy	34 %	1.01
FENNEL	14%	1.10
METIS	8%	1.00

FENNEL

Now we compare FENNEL with Linear Det. Greedy and METIS ($k = 32$):

Name	**RANDOM % edges cut**	**RANDOM part. load**
Linear Det. Greedy	40 %	1.00
FENNEL	14%	1.02
METIS	8%	1.02

FENNEL

... and with METIS alone:

		Fennel		METIS	
m	k	λ	ρ	λ	ρ
7 185 314	4	62.5 %	1.04	65.2%	1.02
6 714 510	8	82.2 %	1.04	81.5%	1.02
6 483 201	16	92.9 %	1.01	92.2%	1.02
6 364 819	32	96.3%	1.00	96.2%	1.02
6 308 013	64	98.2%	1.01	97.9%	1.02
6 279 566	128	98.4 %	1.02	98.8%	1.02

FENNEL

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:

	Run time [s]		Communication [MB]	
# Clusters (k)	Hash	FENNEL	Hash	FENNEL
4	32.27	**25.49**	321.41	**196.9**
8	17.26	**15.14**	285.35	**180.02**
16	10.64	**9.05**	222.28	**148.67**

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.