Minimum Spanning Tree and Connectivity of Large Scale Graphs in MapReduce

Shonan Meeting Seminar
"Large-Scale Distributed Computation"

Donatella Firmani°
joint work with G. Cormode, L. Laura, H. Karloff, S. Muthukrishnan

° Sapienza, University of Rome
Dipartimento di Ingegneria Informatica, Automatica e Gestionale

13 January 2012

Outline

Is there any memory efficient constant round algorithm for connected components in sparse graphs?

Remember yesterday talks by S. Vassilvitskii and S. Lattanzi

- Let us start from computation of MST of Large-Scale graphs
- Map Reduce programming paradigm
- *Semi-External* and *External* Approaches
- Work in Progress and Open Problems ...

Notation Details

Given a weighted undirected graph $G = (V, E)$

- n is the number of vertices
- N is the number of edges
  (size of the input in many MapReduce works)
- all of the edge weights are unique
- G is connected

Sparse Graphs, Dense Graphs and Machine Memory I

(1) SEMI-EXTERNAL MAPREDUCE GRAPH ALGORITHM.
Working memory requirement of any map or reduce computation $O(N^{1-\epsilon})$, for some $\epsilon > 0$

(2) EXTERNAL MAPREDUCE GRAPH ALGORITHM.
Working memory requirement of any map or reduce computation $O(n^{1-\epsilon})$, for some $\epsilon > 0$

Similar definitions for *streaming* and *external memory* graph algorithms

$O(N)$ not allowed!

Sparse Graphs, Dense Graphs and Machine Memory II

(1) G is *dense*, i.e., $N = n^{1+c}$

The design of a semi-external algorithm:

- makes sense for some $\frac{c}{1+c} \geq \epsilon > 0$
  (otherwise it is an external algorithm, $O(N^{1-\epsilon}) = O(n^{1-\epsilon})$)
- allows to store G vertices

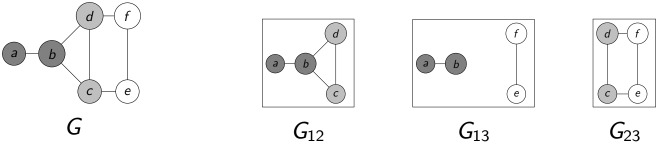
(2) G is *sparse*, i.e., $N = O(n)$

- no difference between semi-external and external algorithms
- storing G vertices is never allowed

Karloff et al. algorithm (SODA '10) I

H. J. Karloff, S. Suri, and S. Vassilvitskii. "A Model of Computation for MapReduce". In: *SODA*. 2010, pp. 938–948

(1) Map Step 1.
Given a number k, randomly partition the set of vertices into k equally sized subsets: $G_{i,j}$ is the subgraph given by $(V_i \cup V_j, E_{i,j})$.



Karloff et al. algorithm (SODA '10) II

(2) REDUCE STEP 1.
For each of the $\binom{k}{2}$ subgraphs $G_{i,j}$, compute the MST (forest) $M_{i,j}$.

---

(3) MAP STEP 2.
Let H be the graph consisting of all of the edges present in some $M_{i,j}$: $H = (V, \bigcup_{i,j} M_{i,j})$: map H to a single reducer \$.

(4) REDUCE STEP 2.
Compute the MST of H.

Karloff et al. algorithm (SODA '10) III

The algorithm is *semi-external*, for dense graphs.

- if G is c-dense and if $k = n^{\frac{c'}{2}}$, for some $c \geq c' > 0$: with high probability, the memory requirement of any map or reduce computation is

$$O(N^{1-\epsilon}) \tag{1}$$

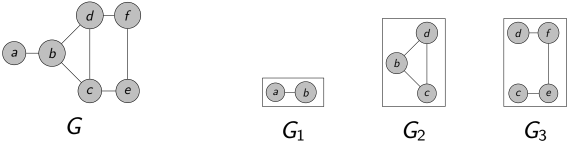
- it works in $2 = O(1)$ rounds

Lattanzi et al. algorithm (SPAA '11) I

S. Lattanzi et al. "Filtering: a method for solving graph problems in MapReduce". In: *SPAA*. 2011, pp. 85–94

(yesterday talk by S. Lattanzi)

(1) MAP STEP i.
Given a number k, randomly partition the set of edges into $\frac{|E|}{k}$ equally sized subsets: G_i is the subgraph given by (V_i, E_i)



Lattanzi et al. algorithm (SPAA '11) II

(2) REDUCE STEP i.
For each of the $\frac{|E|}{k}$ subgraphs G_i, computes the graph G_i', obtained by removing from G_i any edge that is guaranteed not to be a part of any MST because it is the heaviest edge on some cycle in G_i.

Let H be the graph consisting of all of the edges present in some G_i'

- if $|E| \leq k \rightarrow$ the algorithm ends
  (H is the MST of the input graph G)
- otherwise $\rightarrow$ start a new round with H as input

Lattanzi et al. algorithm (SPAA '11) III

The algorithm is *semi-external*, for dense graphs.

- if G is c-dense and if $k = n^{1+c'}$, for some $c \geq c' > 0$:
  the memory requirement of any map or reduce computation is

$$O(n^{1+c'}) = O(N^{1-\epsilon}) \quad (2)$$

for some

$$\frac{c'}{1+c'} \geq \epsilon > 0 \quad (3)$$

- it works in $\lceil \frac{c}{c'} \rceil = O(1)$ rounds

Summary

	[KSV10]	[Lat+11]
	G is c-dense, and $c \geq c' > 0$	
	if $k = n^{\frac{c'}{2}}$, whp	if $k = n^{1+c'}$
Memory	$O(N^{1-\epsilon})$	$O(n^{1+c'}) = O(N^{1-\epsilon})$
Rounds	2	$\lceil \frac{c}{c'} \rceil = O(1)$

Table: Space and Time complexity of algorithms discussed so far.

Experimental Settings (thanks to A. Paolacci)

- **Data Set.**
  Web Graphs, from hundreds of thousand to 7 millions vertices
  `http://webgraph.dsi.unimi.it/`
- **Map Reduce framework.**
  Hadoop 0.20.2 (pseudo-distributed mode)
- **Machine.**
  CPU Intel i3-370M (3M cache, 2.40 Ghz), RAM 4GB, Ubuntu Linux.
- **Time Measures.**
  Average of 10 rounds of the algorithm on the same instance

Preliminary Experimental Evaluation I

Memory Requirement in [KSV10]

	Mb	c	n^{1+c}	$k = n^{1+c'}$	round 1[1]	round 2[1]
cnr-2000	43.4	0.18	3.14	3	7.83	4.82
in-2004	233.3	0.18	3.58	3	50.65	21.84
indochina-2004	2800	0.21	5.26	5	386.25	126.17

Using smaller values of k (decreasing parallelism)

- *decreases* round 1 output size → round 2 time ⌣̈
- *increases* memory and time requirement of round 1 reduce step ⌢̈

[1] output size in Mb

Preliminary Experimental Evaluation II

Impact of Number of Machines in Performances of [KSV10]

	machines	map time (sec)	reduce time (sec)
cnr-2000	1	49	29
cnr-2000	2	44	29
cnr-2000	3	59	29
in-2004	1	210	47
in-2004	2	194	47
in-2004	3	209	52

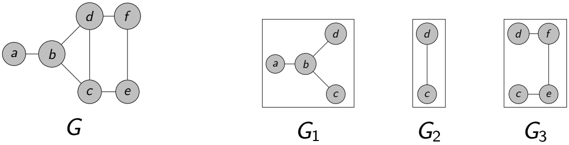
Implications of changes in the number of machines, with $k = 3$: increasing the number of machines *might* increase overall computation time (w.r.t. running more map or reduce instances on the same machine)

Preliminary Experimental Evaluation III

Number of Rounds in [Lat+11]

Let us assume, in the r-th round:

- $|E| > k$;
- each of the subgraphs G_i is a tree or a forest.



input graph = output graph, and the r-th is a “void” round.

Preliminary Experimental Evaluation IV

Number of Rounds in [Lat+11]

(Graph instances having same c value 0.18)

	c'	expected rounds	average rounds[1]
cnr-2000	0.03	8	8.00
cnr-2000	0.05	5	7.33
cnr-2000	0.15	2	3.00
in-2004	0.03	6	6.00
in-2004	0.05	4	4.00
in-2004	0.15	2	2.00

We noticed some few "void" round occurrences.
(Partitioning using a random hash function)

Simulation of PRAMs via MapReduce I

H. J. Karloff, S. Suri, and S. Vassilvitskii. "A Model of Computation for MapReduce". In: *SODA*. 2010, pp. 938–948; Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. "On distributing symmetric streaming computations". In: *ACM Trans. Algorithms* 6 (4 2010), 66:1–66:19; Michael T. Goodrich. "Simulating Parallel Algorithms in the MapReduce Framework with Applications to Parallel Computational Geometry". In: *CoRR* abs/1004.4708 (2010)

(1) CRCW PRAM. via *memory-bound MapReduce* framework.

(2) CREW PRAM. via $\mathcal{DMRC}$:
(PRAM) $O(S^{2-2\epsilon})$ total memory, $O(S^{2-2\epsilon})$ processors and T time.
(MapReduce) $O(T)$ rounds, $O(S^{2-2\epsilon})$ reducer instances.

(3) EREW PRAM. via MUD model of computation.

PRAM Algorithms for the MST

- CRCW PRAM algorithm [Cole, Klein, and Tarjan [CKT96]] (randomized)
  $O(\log n)$ time, $O(N)$ work $\rightarrow$ work-optimal
- CREW PRAM algorithm [JáJá [JáJ92]]
  $O(\log^2 n)$ time, $O(n^2)$ work $\rightarrow$ work-optimal if $N = O(n^2)$.
- EREW PRAM algorithm [Johnson and Metaxas [JM92]]
  $O(\log^{\frac{3}{2}} n)$ time, $O(N \log^{\frac{3}{2}} n)$ work.
- EREW PRAM algorithm [Pettie and Ramachandran [PR02]] (randomized)
  $O(N)$ total memory, $O(\frac{N}{\log n})$ processors.
  $O(\log n)$ time, $O(N)$ work $\rightarrow$ work-time optimal.

Simulation of CRCW PRAM with CREW PRAM: $\Omega(\log S)$ steps.

Simulation of [PR02] via MapReduce I

The algorithm is *external* (for dense and sparse graphs).

Simulate the algorithm in [PR02] using CREW→MapReduce.

- the memory requirement of any map or reduce computation is

$$O(\log n) = O(n^{1-\epsilon}) \tag{4}$$

for some

$$1 - \log \log n \geq \epsilon > 0 \tag{5}$$

- the algorithm works in $O(\log n)$ rounds.

Summary

	[KSV10]	[Lat+11]	Simulation
	G is c-dense, and $c \geq c' > 0$		
	if $k = n^{\frac{c'}{2}}$, whp	if $k = n^{1+c'}$	
Memory	$O(N^{1-\epsilon})$	$O(n^{1+c'}) = O(N^{1-\epsilon})$	$O(\log n) = O(n^{1-\epsilon})$
Rounds	2	$\lceil \frac{c}{c'} \rceil = O(1)$	$O(\log n)$

Table: Space and Time complexity of algorithms discussed so far.

Borůvka MST algorithm I

O. Borůvka. "O jistém problému minimálním (About a Certain Minimal Problem)". In: III (1926), 37–58

Classical model of computation algorithm

procedure `Borůvka MST`$(G(V, E))$:
$T \rightarrow V$
while $|T| < n - 1$ **do**
  for all connected component C in T **do**
    $e \rightarrow$ the smallest-weight edge from C to another component in T
    if $e \notin T$ **then**
      $T \rightarrow T \cup \{e\}$
    end if
  end for
end while

Borůvka MST algorithm II

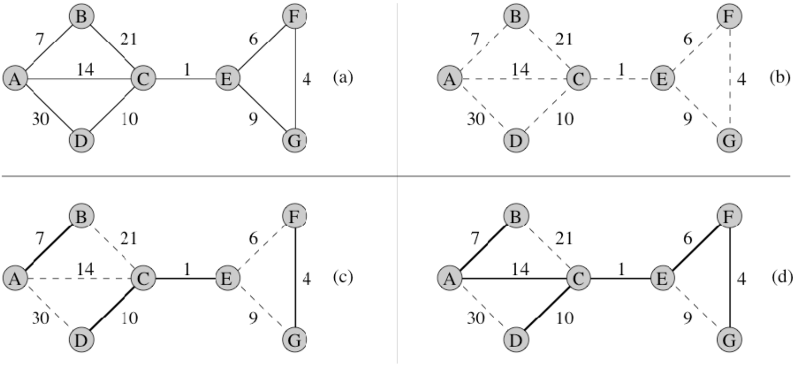


Figure: An example of Borůvka algorithm execution.

Random Mate CC algorithm I

Hillel Gazit. "An Optimal Randomized Parallel Algorithm for Finding Connected Components in a Graph". In: *SIAM Journal on Computing* 20.6 (1991), pp. 1046–1067

CRCW PRAM model of computation algorithm

procedure `Random Mate CC`($G(V,E)$):
for all $v \in V$ **do** $cc(v) \rightarrow v$ **end for**
while there are edges connecting two CC in G (*live*) **do**
  for all $v \in V$ **do** gender[v] $\rightarrow$ rand({M, F}) **end for**
  for all live $(u,v) \in V$ **do**
    $cc(u)$ is M $\wedge$ $cc(v)$ is F ? $cc(cc(u)) \rightarrow cc(v)$: $cc(cc(v)) \rightarrow cc(u)$
  end for
  for all $v \in E$ **do** $cc(v) \rightarrow cc(cc(v))$ **end for**
end while

Random Mate CC algorithm II

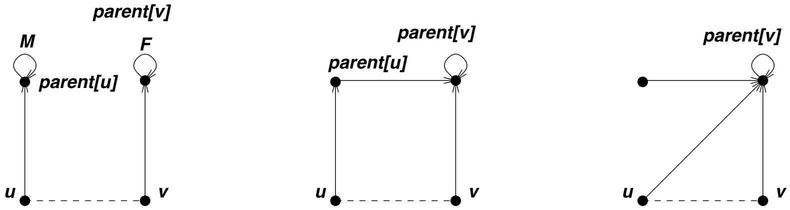
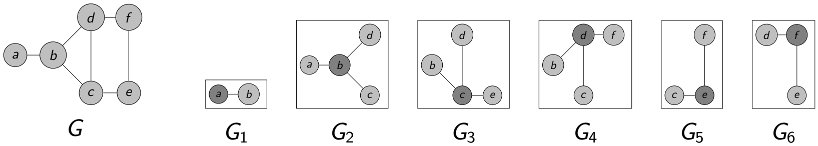


Figure: An example of Random Mate algorithm step.

Borůvka + Random Mate I

Let us consider again the labeling function $cc : V \to V$

(1) MAP STEP i (BORŮVKA).
Given an edge $(u, v) \in E$, the result of the mapping consists in two *key* : *value* pairs $cc(u) : (u, v)$ and $cc(v) : (u, v)$.



Borůvka + Random Mate II

(2) REDUCE STEP i (BORŮVKA).
For each subgraph G_i, execute one iteration of the Borůvka algorithm.

Let T be the output of i-th Borůvka iteration.
Execute r_i Random Mate rounds, feeding the first one with T.

(3) ROUND $i + j$ (RANDOM MATE).
Use a MapReduce implementation [Piccolboni [Pic10]] of Random Mate algorithm and update the function cc.

- if there are no more live edges, the algorithm ends (T is the MST of the input graph G)
- otherwise $\rightarrow$ start a new Borůvka round

Borůvka + Random Mate III

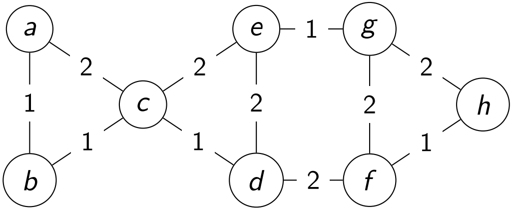
Two extremal cases:

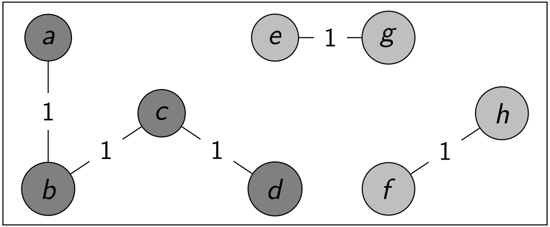
- output of first Borůvka round is connected
  $\rightarrow O(\log n)$ Random Mate rounds, and algorithm ends.
- output of each Borůvka round is a matching
  $\rightarrow \forall i, r_i = 1$ Random Mate round
  $\rightarrow O(\log n)$ Borůvka rounds, and algorithm ends.

Therefore

- it works in $O(\log^2 n)$ rounds;
- example working in $\approx \frac{1}{4} \log^2 n$

Borůvka + Random Mate IV





Conclusions

Work in progress for an *external* implementation of the algorithm (for dense and sparse graphs).

- the worst case seems to rely on a certain kind of structure in the graph, difficult to appear in realistic graphs
- need of more experimental work to confirm it

Is there any external constant round algorithm for connected components and MST in sparse graphs?

Maybe under certain (and hopefully realistic) assumptions.

THANK YOU

:)