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
Introduction

Map Reduce Algorithms

Simulating PRAM Algorithms

Borůvka + Random Mate
Karloff et al. algorithm (SODA ’10) I


(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 $(V_i \cup V_j, E_{i,j})$. 

![Graphs G, G_{12}, G_{13}, G_{23}](image-url)
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

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

- it works in $2 = O(1)$ rounds
Lattanzi et al. algorithm (SPAA ’11)!

(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

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

  (2)

  for some

  $$\frac{c'}{1+c'} \geq \epsilon > 0$$

  (3)

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

<table>
<thead>
<tr>
<th>Memory</th>
<th>Space Complexity</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>[KSV10]</td>
<td>$G$ is $c$-dense, and $c \geq c' &gt; 0$</td>
<td>$O(N^{1-\epsilon})$</td>
</tr>
<tr>
<td>[Lat+11]</td>
<td>if $k = n^{\frac{c'}{2}}$, whp</td>
<td>$\frac{c}{c'} = O(1)$</td>
</tr>
<tr>
<td></td>
<td>if $k = n^{1+c'}$</td>
<td>$O(n^{1+c'}) = O(N^{1-\epsilon})$</td>
</tr>
</tbody>
</table>

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]

<table>
<thead>
<tr>
<th></th>
<th>Mb</th>
<th>c</th>
<th>$n^{1+c}$</th>
<th>$k = n^{1+c'}$</th>
<th>round 1</th>
<th>round 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnr-2000</td>
<td>43.4</td>
<td>0.18</td>
<td>3.14</td>
<td>3</td>
<td>7.83</td>
<td>4.82</td>
</tr>
<tr>
<td>in-2004</td>
<td>233.3</td>
<td>0.18</td>
<td>3.58</td>
<td>3</td>
<td>50.65</td>
<td>21.84</td>
</tr>
<tr>
<td>indochina-2004</td>
<td>2800</td>
<td>0.21</td>
<td>5.26</td>
<td>5</td>
<td>386.25</td>
<td>126.17</td>
</tr>
</tbody>
</table>

Using smaller values of $k$ (decreasing parallelism)

- decreases round 1 output size $\rightarrow$ round 2 time $\nearrow$
- increases memory and time requirement of round 1 reduce step $\nearrow$

[1] output size in Mb
Preliminary Experimental Evaluation II

Impact of Number of Machines in Performances of [KSV10]

<table>
<thead>
<tr>
<th></th>
<th>machines</th>
<th>map time (sec)</th>
<th>reduce time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnr-2000</td>
<td>1</td>
<td>49</td>
<td>29</td>
</tr>
<tr>
<td>cnr-2000</td>
<td>2</td>
<td>44</td>
<td>29</td>
</tr>
<tr>
<td>cnr-2000</td>
<td>3</td>
<td>59</td>
<td>29</td>
</tr>
<tr>
<td>in-2004</td>
<td>1</td>
<td>210</td>
<td>47</td>
</tr>
<tr>
<td>in-2004</td>
<td>2</td>
<td>194</td>
<td>47</td>
</tr>
<tr>
<td>in-2004</td>
<td>3</td>
<td>209</td>
<td>52</td>
</tr>
</tbody>
</table>

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)
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

<table>
<thead>
<tr>
<th></th>
<th>c'</th>
<th>expected rounds</th>
<th>average rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnr-2000</td>
<td>0.03</td>
<td>8</td>
<td>8.00</td>
</tr>
<tr>
<td>cnr-2000</td>
<td>0.05</td>
<td>5</td>
<td>7.33</td>
</tr>
<tr>
<td>cnr-2000</td>
<td>0.15</td>
<td>2</td>
<td>3.00</td>
</tr>
<tr>
<td>in-2004</td>
<td>0.03</td>
<td>6</td>
<td>6.00</td>
</tr>
<tr>
<td>in-2004</td>
<td>0.05</td>
<td>4</td>
<td>4.00</td>
</tr>
<tr>
<td>in-2004</td>
<td>0.15</td>
<td>2</td>
<td>2.00</td>
</tr>
</tbody>
</table>

We noticed some few “void” round occurrences.

(Graph instances having same c value 0.18)
Introduction

Map Reduce Algorithms

Simulating PRAM Algorithms

Borůvka + Random Mate
Introduction

Map Reduce Algorithms

Simulating PRAM Algorithms

Borůvka + Random Mate

Simulation of PRAMs via MapReduce I


(1) CRCW PRAM. via memory-bound MapReduce framework.
(2) CREW PRAM. via 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]]
  $O(\log^{3/2} n)$ time, $O(N \log^{3/2} n)$ 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

\[ O(\log n) = O(n^{1-\epsilon}) \]  \hspace{1cm} (4)

for some

\[ 1 - \log \log n \geq \epsilon > 0 \]  \hspace{1cm} (5)

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

<table>
<thead>
<tr>
<th>Memory</th>
<th>[KSV10]</th>
<th>[Lat+11]</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rounds</td>
<td>$G$ is $c$-dense, and $c \geq c' &gt; 0$</td>
<td>$k = n^{c' \over 2}$, whp</td>
<td>$O(\log n) = O(n^{1-\epsilon})$</td>
</tr>
<tr>
<td></td>
<td>$O(N^{1-\epsilon})$</td>
<td>$k = n^{1+c'}$</td>
<td>$O(n^{1+c'}) = O(N^{1-\epsilon})$</td>
</tr>
<tr>
<td></td>
<td>$2$</td>
<td>$\lceil c' \over c \rceil = O(1)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

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

Map Reduce Algorithms

Simulating PRAM Algorithms

Borůvka + Random Mate
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

```plaintext
procedure Borůvka MST(G(V, E)):
    T → V
    while |T| < n − 1 do
        for all connected component C in T do
            e → the smallest-weight edge from C to another component in T
            if e ∉ T then
                T → T ∪ {e}
            end if
        end for
    end while
```
Borůvka MST algorithm II

Figure: An example of Borůvka algorithm execution.
Random Mate CC algorithm I


CRCW PRAM model of computation algorithm

```plaintext
procedure Random Mate CC(G(V, E)):
    for all v ∈ V do cc(v) → v end for
    while there are edges connecting two CC in G (live) do
        for all v ∈ V do gender[v] → rand({M, F}) end for
        for all live (u, v) ∈ V do
            cc(u) is M ∧ cc(v) is F ? cc(cc(u)) → cc(v) : cc(cc(v)) → cc(u)
        end for
        for all v ∈ E do cc(v) → cc(cc(v)) end for
    end while
```
Random Mate CC algorithm II

Figure: An example of Random Mate algorithm step.
Let us consider again the labeling function $cc : V \rightarrow V$.

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 $cc(v) : (u, v)$. 

---

$G$  

$G_1$  

$G_2$  

$G_3$  

$G_4$  

$G_5$  

$G_6$
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

Graph representation: A network of nodes connected by edges with weights.

Mathematical notation: Formulas related to the algorithm.

Diagram: A visual representation of the algorithm's execution steps.

Legend: Descriptions of specific elements or steps in the algorithm.
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

😊