Cloud Computing

Link Analysis in the Cloud

Dell Zhang
Birkbeck, University of London
2018/19
Graph Problems & Representations
What is a Graph?

• $G = (V,E)$, where
  – $V$ represents the set of vertices (nodes)
  – $E$ represents the set of edges (links)
  – Both vertices and edges may contain additional information (e.g., edge weights)

• Different types of graphs:
  – directed vs. undirected edges
  – presence or absence of cycles
We See Graphs Everywhere

• Ubiquitous network (graph) data
  • Technological Network
    • Internet
  • Information Network
    • WWW, Sematic Web/Ontologies, XML/RD
    – Social network
    – Biological Network
    – Financial Network
    – Transportation Network
Some Graph Problems

• Finding shortest paths
  – Routing Internet traffic and UPS trucks

• Finding minimum spanning trees
  – Telecommunication companies laying down fibre

• Finding max flow
  – Airline scheduling
Some Graph Problems

• Identify “special” nodes and communities
  – Breaking up terrorist cells, spread of avian flu
• Bipartite matching
  – Monster.com, Match.com
• And of course... PageRank
Challenge in Dealing with Graph Data

• Flat Files
  – No query support

• RDBMS
  – Can store the graph
  – But limited support for graph query
    • Connect-By (Oracle)
    • Common Table Expressions (CTEs) (Microsoft)
    • Temporal Table
Native Graph Databases

• An Emerging Field

• Storage and Basic Operators
  – Neo4j (an open source graph database), InfiniteGraph, VertexDB, ...

• Distributed Graph Processing (mostly in-memory-only)
  – Google’s Pregel, GraphLab, ...
The Graph Analytics Industry

• Status of Practice
  – Graph data in many industries
  – Graph analytics are powerful and can bring great business values/insights
  – Graph analytics not utilized enough in enterprises due to lack of available platforms/tools (except leading tech companies which have high caliber in house engineering teams and resources)
Graphs and MapReduce

• Graph algorithms typically involve:
  – Performing computations at each node: based on node features, edge features, and local link structure
  – Propagating computations: “traversing” the graph

• Key questions:
  – How do you represent graph data in MapReduce?
  – How do you traverse a graph in MapReduce?
Representing Graphs

• Two common representations
  – Adjacency matrix
  – Adjacency list
Adjacency Matrices

- Represent a graph as an $n \times n$ square matrix $M$
  - $n = |V|$
  - $M_{ij} = 1$ means a link from node $i$ to $j$

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Adjacency Matrices: Critique

• Advantages:
  – Amenable to mathematical manipulation
  – Iteration over rows and columns corresponds to computations on out-links and in-links

• Disadvantages:
  – Lots of zeros for sparse matrices
  – Lots of wasted space
## Adjacency Lists

- Take adjacency matrices... and throw away all the zeros

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1: 2, 4  
2: 1, 3, 4  
3: 1  
4: 1, 3
Adjacency Lists: Critique

• Advantages:
  – Much more compact representation
  – Easy to compute over out-links

• Disadvantages:
  – Much more difficult to compute over in-links
Parallel Breadth-First Search
Single Source Shortest Path

• **Problem:** find shortest path from a source node to one or more target nodes
  – “shortest” might also mean lowest weight or cost

• First, a refresher: Dijkstra’s algorithm
Dijkstra’s Algorithm

Example from CLR
Dijkstra’s Algorithm

Example from CLR
Dijkstra’s Algorithm

Example from CLR
Dijkstra’s Algorithm

Example from CLR
Dijkstra’s Algorithm

Example from CLR
Dijkstra’s Algorithm

Example from CLR
Single Source Shortest Path

**Problem:** find shortest path from a source node to one or more target nodes
- “shortest” might also mean lowest weight or cost

**On a single machine:** Dijkstra’s algorithm

**MapReduce:** Parallel Breadth-First Search (BFS)
- Consider simplest case of equal edge weights first
- Solution to the problem can be defined inductively
Finding the Shortest Path

Here’s the intuition:

– Define: \( b \) is reachable from \( a \) if \( b \) is in the adjacency list of \( a \)
– \( \text{DISTANCE}(s) = 0 \)
– For all nodes \( p \) reachable from \( s \):
  \( \text{DISTANCE}(p) = 1 \)
– For all nodes \( n \) reachable from some other set of nodes \( M \):
  \( \text{DISTANCE}(n) = 1 + \min_{m \in M} \text{DISTANCE}(m) \)
Finding the Shortest Path
Visualizing Parallel BFS
From Intuition to Algorithm

• Data representation:
  – Key: node \( n \)
  – Value: 
    \( d \) (distance from start), adjacency list (list of nodes reachable from \( n \))
  – Initialization:
    for all nodes except the start node, \( d = \infty \).
From Intuition to Algorithm

• Mapper:
  – \( \forall m \in \text{adjacency list}: \text{emit } (m, \, d + 1) \)

• Sort/Shuffle
  – Groups distances by reachable nodes

• Reducer:
  – Selects the minimum distance path for each reachable node
  – Additional bookkeeping needed to keep track of the actual path
Multiple Iterations Needed

• Each MapReduce iteration advances the “known frontier” by one hop
  – Subsequent iterations include more and more reachable nodes as frontier expands
  – Multiple iterations are needed to explore entire graph

• Preserving graph structure:
  – Problem: Where did the adjacency list go?
  – Solution: mapper emits \((n, \text{adjacency list})\) as well
BFS Pseudo-Code

1: class Mapper
2:   method MAP(nid n, node N)
3:       d ← N.DISTANCE
4:       EMIT(nid n, N) ▶ Pass along graph structure
5:       for all nodeid m ∈ N.ADJACENCYLIST do
6:           EMIT(nid m, d + 1) ▶ Emit distances to reachable nodes

1: class Reducer
2:   method REDUCE(nid m, [d₁, d₂, . . .])
3:       d_min ← ∞
4:       M ← ∅
5:       for all d ∈ counts [d₁, d₂, . . .] do
6:           if ISNODE(d) then
7:               M ← d ▶ Recover graph structure
8:           else if d < d_min then ▶ Look for shorter distance
9:               d_min ← d
10:              M.DISTANCE ← d_min ▶ Update shortest distance
11:             EMIT(nid m, node M)
Stopping Criterion

• How many iterations are needed in parallel BFS (equal edge weight case)?
• Convince yourself: when a node is first “discovered”, we’ve found the shortest path
• Now answer the question...
  – Six degrees of separation?
• Practicalities of implementation in MapReduce
Weighted Edges

• Now add positive weights to the edges
  – Why can’t edge weights be negative?
• Simple change: adjacency list now includes a weight $w$ for each edge
  – In mapper, emit $(m, d + w_p)$ instead of $(m, d + 1)$ for each node $m$
• That’s it?
Stopping Criterion

• How many iterations are needed in parallel BFS (positive edge weight case)?

• Convince yourself: when a node is first “discovered”, we’ve found the shortest path

\textit{Not true!}
How many iterations are required to discover the shortest distances to all nodes from $n_1$?
Stopping Criterion

• How many iterations are needed in parallel BFS (positive edge weight case)?
• Practicalities of implementation in MapReduce
Comparison to Dijkstra

• Dijkstra’s algorithm is more efficient
  – At any step it only pursues edges from the minimum-cost path inside the frontier

• MapReduce explores all paths in parallel
  – Lots of “waste”
  – Useful work is only done at the “frontier”

• Why can’t we do better using MapReduce?
Implementation on Hadoop

http://goo.gl/TEoU4
Graphs and MapReduce

• Generic recipe:
  – Represent graphs as adjacency lists
  – Perform local computations in mapper
  – Pass along partial results via out-links, keyed by destination node
  – Perform aggregation in reducer on in-links to a node
  – Iterate until convergence: controlled by external “driver”
  – Don’t forget to pass the graph structure between iterations
PageRank

Diagram showing the concept of PageRank with various smiling faces connected by arrows.
Random Walks over the Web

• Random surfer model:
  – User starts at a random Web page
  – User randomly clicks on links, surfing from page to page

• PageRank
  – Characterizes the amount of time spent on any given page
  – Mathematically, a probability distribution over pages
Random Walks over the Web

• PageRank captures the notion of page importance
  – Correspondence to human intuition?
  – One of thousands of features used in Web search
  – Note: query-independent
PageRank: Simplified
PageRank: Simplified

• Given page $x$ with in-links $t_1 \ldots t_n$, where
  – $C(t)$ is the out-degree of $t$

$$PR(x) = \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$
Example: the Web in 1839

Yahoo

Amazon

Microsoft

\[
\begin{array}{ccc}
y & a & m \\
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 1 \\
m & 0 & 1/2 & 0 \\
\end{array}
\]
Simulating a Random Walk

• Start with the vector $\mathbf{v} = [1,1,\ldots,1]$ representing the idea that each Web page is given one unit of \textit{importance}.

• Repeatedly apply the matrix $M$ to $\mathbf{v}$, allowing the importance to flow like a random walk.

• Limit exists, but about 50 iterations is sufficient to estimate final distribution.
Example: the Web in 1839

- Equations $v = M \cdot v$:

  \[
  y = y / 2 + a / 2 \\
  a = y / 2 + m \\
  m = a / 2
  \]

\[
\begin{array}{ccccccc}
  y & 1 & 1 & 5/4 & 9/8 & \cdots & 6/5 \\
  a & 1 & 3/2 & 1 & 11/8 & \cdots & 6/5 \\
  m & 1 & 1/2 & 3/4 & 1/2 & & 3/5 \\
\end{array}
\]
Solving the Equations

- Because there are no constant terms, these 3 equations in 3 unknowns do not have a unique solution.
- Add in the fact that $y + a + m = 3$ to solve.
- In Web-sized examples, we cannot solve by Gaussian elimination, but we need to use the power method (= iterative solution).
Computing PageRank

• Properties of PageRank
  – Can be computed iteratively
  – Effects at each iteration are local
Computing PageRank

• Sketch of algorithm:
  – Start with seed $PR_i$ values
  – Each page distributes its $PR_i$ “credit” to all of its out-links
  – Each page adds up the “credits” from all of its in-links to compute $PR_{i+1}$
  – Iterate until the values converge
Sample PageRank Iterations

**Iteration 1**
Sample PageRank Iterations

Iteration 2

- $n_1 (0.066)$
- $n_2 (0.166)$
- $n_3 (0.166)$
- $n_4 (0.3)$
- $n_5 (0.3)$

- $n_1 (0.1)$
- $n_2 (0.133)$
- $n_3 (0.183)$
- $n_4 (0.2)$
- $n_5 (0.383)$

- Arrows and weights between nodes represent the transition probabilities in the PageRank algorithm.
PageRank in MapReduce
PageRank Pseudo-Code

1: class Mapper
2:     method Map(nid n, node N)
3:         \[ p \leftarrow N.PAGERANK / |N.ADJACENCYLIST| \]
4:         Emit(nid n, N) \quad \triangleright \text{Pass along graph structure}
5:     for all nodeid \( m \in N.ADJACENCYLIST \) do
6:         Emit(nid m, p) \quad \triangleright \text{Pass PageRank mass to neighbors}

1: class Reducer
2:     method Reduce(nid m, [p_1, p_2, \ldots])
3:         M \leftarrow \emptyset
4:     for all \( p \in \text{counts} [p_1, p_2, \ldots] \) do
5:         if \text{IsNode}(p) then
6:             M \leftarrow p \quad \triangleright \text{Recover graph structure}
7:         else
8:             s \leftarrow s + p \quad \triangleright \text{Sum incoming PageRank contributions}
9:     M.PAGERANK \leftarrow s
10:    Emit(nid m, node M)
Real-World Problems

• Some pages are “dead ends” (no out-links).
  – Such a page causes importance to leak out.

• Some other (groups of) pages are *spider traps* (all out-links are within the group).
  – Eventually spider traps absorb all importance.
Microsoft becomes a dead end
Microsoft becomes a dead end

• Equations $v = M v$:
  
  $y = y/2 + a/2$
  
  $a = y/2$
  
  $m = a/2$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>3/4</th>
<th>5/8</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>1</td>
<td>1</td>
<td>3/4</td>
<td>5/8</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>1/2</td>
<td>1/2</td>
<td>3/8</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>$m$</td>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Microsoft becomes a spider trap

Amazon

Microsoft

Yahoo

\[
\begin{pmatrix}
y & a & m \\
y & 1/2 & 1/2 & 0 \\
a & 1/2 & 0 & 0 \\
m & 0 & 1/2 & 1 \\
\end{pmatrix}
\]
Microsoft becomes a spider trap

• Equations \( \mathbf{v} = \mathbf{M} \mathbf{v} : \)

\[
\begin{align*}
y &= y/2 + a/2 \\
a &= y/2 \\
m &= a/2 + m 
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>3/4</th>
<th>5/8</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1/2</td>
<td>1/2</td>
<td>3/8</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>3/2</td>
<td>7/4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Google’s Solution

• “Tax” each page a fixed percentage at each iteration.
• Add the same constant to all pages.
• Models a random walk with a fixed probability of going to a random place next.
Example: with 20% Tax

• Equations \( v = 0.8(M \ v) + 0.2 \):

\[
\begin{align*}
y &= 0.8(y /2 + a/2) + 0.2 \\
a &= 0.8(y /2) + 0.2 \\
m &= 0.8(a /2 + m) + 0.2
\end{align*}
\]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1</td>
<td>1.00</td>
<td>0.84</td>
<td>0.776</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>0.60</td>
<td>0.60</td>
<td>0.536</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>1.40</td>
<td>1.56</td>
<td>1.688</td>
</tr>
</tbody>
</table>
PageRank: Complete

• Two additional complexities
  – What is the proper treatment of dangling nodes (i.e., nodes with no out-links)?
  – How do we factor in the random jump factor?
PageRank: Complete

• Solution:
  – Second pass to redistribute “missing PageRank mass” and account for random jumps

  \[
  p' = \alpha \left( \frac{1}{N} \right) + (1 - \alpha) \left( \frac{m}{N} + p \right)
  \]

• \( p \) is PageRank value from before,
  \( p' \) is updated PageRank value

• \( N \) is the total number of nodes in the graph

• \( m \) is the missing PageRank mass
PageRank Convergence

• Alternative convergence criteria
  – Iterate until PageRank values don’t change
  – Iterate until PageRank rankings don’t change
  – Fixed number of iterations

• Convergence for web graphs?
Beyond PageRank

• Link structure is important for web search
  – PageRank is one of many link analysis algorithms: HITS, SALSA, etc.
  – Used with thousands of other features in ranking...

• Adversarial nature of web search
  – Link spamming
  – Spider traps
  – Keyword stuffing
  – ...

Efficient Graph Algorithms

- Sparse vs. Dense Graphs
- Graph Topologies
Local Aggregation

• Use combiners!
  – In-mapper combining design pattern also applicable

• Maximize opportunities for local aggregation
  – Simple tricks: sorting the dataset in specific ways
Take Home Messages

• Graph Problems and Representations
• Parallel Breadth-First Search
• PageRank: Simplified and Complete