大图数据管理中的关键算法

肖仰华

GDM@FUDAN (http://gdm.fudan.edu.cn)

复旦大学
Our Focus

Key graph algorithms on various, large scale real graphs.
Challenges caused by real graphs

Key algorithms in managing big graphs
Real Graphs are Big

Scaling up to big graphs is challenging!

Key algorithms in managing big graphs
Real Graphs are Complex

- Skewed degree distribution
- Unclear community structure
- Small world

Key algorithms in managing big graphs
Skewed degree distribution

Balanced load distribution is challenging!
Small world

• Six-degree separation
• Even smaller on social media network
• Graphs are becoming denser (kleinbeger 07)

Exploration in graphs is costly!
Unclear community structure

Newman 2006

<table>
<thead>
<tr>
<th>Network</th>
<th>Size $n$</th>
<th>GN</th>
<th>CNM</th>
<th>DA</th>
<th>Modularity $Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karate</td>
<td>34</td>
<td>0.401</td>
<td>0.381</td>
<td>0.419</td>
<td>0.419</td>
</tr>
<tr>
<td>Jazz musicians</td>
<td>198</td>
<td>0.405</td>
<td>0.439</td>
<td>0.445</td>
<td>0.442</td>
</tr>
<tr>
<td>Metabolic</td>
<td>453</td>
<td>0.403</td>
<td>0.402</td>
<td>0.434</td>
<td>0.435</td>
</tr>
<tr>
<td>E-mail</td>
<td>1,133</td>
<td>0.532</td>
<td>0.494</td>
<td>0.574</td>
<td>0.572</td>
</tr>
<tr>
<td>Key signing</td>
<td>10,680</td>
<td>0.816</td>
<td>0.733</td>
<td>0.846</td>
<td>0.855</td>
</tr>
<tr>
<td>Physicists</td>
<td>27,519</td>
<td>—</td>
<td>0.668</td>
<td>0.679</td>
<td>0.723</td>
</tr>
</tbody>
</table>

Poor data parallelism: Partitioning a graph is hard!
Real Graphs have Complex Operations

• Online query processing
  – Shortest distance/path query
  – Subgraph matching query
  – SPARQL query
  – ...

• Offline graph analytics
  – Ranking: PageRank, SImRank
  – Clustering
  – Classification
  – Community detection
  – ...

• Other graph operations
  – Graph generation, visualization, interactive exploration, etc.

Graph computations are diversified!
Graph Operations need Random Access

• Data access along edges
  – Accessing a node’s neighbor requires “jumping” on graphs.
  – Consider DFS traverse
    • Hard to employ the locality of the graph

• Random access on disk or network is costly!
Order-dependent logic

• Process vertex in a dynamic order
  – Consider Dijkstra algorithm,
    • Always select the unvisited vertex that has the minimal weight to the visited nodes
  – Consider DFS
    • The next vertex to be visited is dependent on the previously visited node sequence

• Poor computation parallelism!
• Hard to be parallelized!
Current Status

• Good solutions for small or intermediate size (million-nodes) graph:
  – We need to process billion-node graphs

• Good solution for serial algorithms
  – We need parallelized solutions

• Good solution for regular graphs
  – We need to handle irregular graphs
Taming big graph processing

• Our solution
  – Memory cloud: Trinity
  – Leverage properties of real graphs
    • Scale free
    • Small world
    • Community structure
  – Lightweight model and algorithm
    • Clustering instead of partitioning
    • Approximate distance oracle instead of accurate shortest path query
    • Repartitioning instead of direct partitioning

Why not hadoop and MapReduce?

- Not suitable for online processing
- Huge Disk/Network IO, not suitable for random access on graphs
- Only algorithms fit for vertex-centric framework can achieve good performance on carefully-designed systems: such as Pregel

Key algorithms in managing big graphs
Outline

- Graph is big
  - billion node graphs
- Graph is distributed
  - Instead of centrically stored
- Solutions should be general enough to run on a general-purpose graph system
Graph partitioning
Graph Partitioning

• Problem definition
  – Divide a graph into $k$ almost equal-size parts, such that the number of edges among them is minimized.
  – Weighted variance

• Why?
  – Load balance
  – Reduce communication

• Example: BFS on the graph
  – Best partitioning needs 3 remote accesses
  – Worst partitioning needs 17

• NP-complete [Garey74]
Existing solutions

- Classical partitioning algorithm
  - Swapping selected node pairs: KL [kernighan 72] and FM [Fiduccia 82]
  - Simulated annealing based solutions [Johnson89]
  - Genetic algorithms [Bui96]
  - For small graphs (but with high quality)
- Multi-level partitioning solutions
  - METIS [Karypis95], Chaco [Hendrickson] and Scotch [Pellegrini96]
  - For million-node graphs
- Parallelized Multi-level partitioning solutions
  - ParMetis [Karypis98] and Pt-Scotch [Chevalier08].
  - For at most tens-of-million-node graphs.
- No good solutions for partitioning billion-node graphs on a general-purpose distributed system.
Graph partitioning in current systems

• Default: *Random partitioning*
  – PBGL, Pregel, neo4j, InfiniteGraph, HyperGraphDB
  – Quality is significantly worse than a refined method

• No partitioning algorithm is supported at the system level

• PBGL and Pregel support *user-defined partitioning*
KL

- Iteratively bisect a graph into two parts with equal size so that the number of cross edges is minimized

- In each bisection
  - Heuristically select the vertex pair whose swap leads to the most gain

- The gain of to swap two vertices a and b that lie in different parts,
  - \( g(a,b) = D_a + D_b - c_{a,b} \)
  - \( D_x = E_x - I_x \)
  - \( E_x \) is the number of neighbors of \( x \) on the different machine
  - \( I_x \) is the number of neighbors of \( x \) on the same machine
  - \( c_{a,b} \) is 1 iff \( a \) and \( b \) are neighbors.
State-of-art method: Metis

- A multi-level framework
- Three phases
  - Coarsening by **maximal match** until the graph is small enough
  - Partition the coarsest graph by KL algorithm [kernighan 72]
  - Uncoarsening

Ref: [Metis 1995]
Maximal Matching

- A **matching** of a graph $G(N,E)$ is a subset $E_m$ of $E$ such that no two edges in $E_m$ share an endpoint.

- A **maximal matching** of a graph $G(N,E)$ is a matching $E_m$ to which no more edges can be added and remain a matching.

- Simple greedy algorithm
Maximal Matching - Example

How to coarsen a graph using a maximal matching

\[ G = (N, E) \]

\( E_m \) is shown in red

Edge weights shown in blue

Node weights are all one

\[ G_c = (N_c, E_c) \]

\( N_c \) is shown in red

Edge weights shown in blue

Node weights shown in black

From James Demmel, [http://www.cs.berkeley.edu/~demmel/cs267_Spr99](http://www.cs.berkeley.edu/~demmel/cs267_Spr99)

33th CCF advanced seminar, Beijing, 2012-11-16
Coarsening in Metis

- Assumption of coarsening: An optimal partitioning on a coarser graph is a good partitioning in the finer graph

- It holds *only when node degree is bounded* (2D, 3D meshes).

- But real networks have *skewed degree distribution*

- Metis uses refinement in the uncoarsening
Possible solution: Multi-level Label Propagation

- Coarsening by label propagation
  - In each iteration, a vertex takes the label that is prevalent in its neighborhood as its own label.

- Lightweight
  - Easily implemented by message passing
  - Easily parallelizable

- Effective (for real networks)
  - Capable of discovering inherent community structures
How to handle imbalance?

- Imbalance caused by label propagation
  - Too many small clusters
  - Some extremely large clusters

- Possible solution:
  - First, limit the size of each cluster
  - Second, merge small clusters by
    - multiprocessor scheduling (MS)
    - weighted graph partitioning (WGP)

- For details:
  - How to partition a billion node graph, Microsoft Technical Report, 2012
New partitioning model on big graphs

• Repartitioning
  – Input: an existing partitioning and a graph
  – Output: a new partitioning with minimal edge cut that overlap significantly with the given partitioning

• Partitioning with replication
  – A balanced graph a partitioning but allowing overlapping, with constraint on the maximal number of replicated vertices
  – Side-effect: inconsistency

• Streaming graph partitioning (SGP)
  – Graph comes as streams
  
Ref: Feng 2012, Distributed Big Graph Storage, CCF Communications,
Greedy solution of SGP

- For every incoming vertex, select one of the machine that can maximize the gain function.

- Let v be the incoming vertex, the gain function of machine m is:
  - \( F(m) = a(m) d(v) \)
  - Quality: \( d(v) \) is the number of v’s neighbors in m
  - Balance: \( a(m) \) is a punishing function of m’s used capacity p
    - Linear function: \( 1 - \frac{p}{c} \)
    - Exponential function: \( 1 - \exp(p-c) \)

- Deterministic greedy selection with a linear \( a(m) \) practically produces the best partitioning (Stanton KDD 2012)
Distance Oracle
Why shortest distance query

- A basic graph operator
  - Used in many graph algorithms, including computing centrality, betweenness, etc.

- What is your Erdos number?
  - The shortest distance from you to mathematician Paul Erdos in the scientific collaboration network
  - Shortest distance can also be used to compute centrality, betweenness
Why distance oracle?

• Exact solutions
  – Online computation
    • Dijkstra-like algorithms, BFS
    • At least linear complexity, computational prohibitive on large graphs
  – Pre-compute all pairs of shortest path
    • Of quadratic space complexity

• Distance oracle
  – Report *approximate* distance between any two vertices in a graph in constant time by *pre-computation*
  – When graph is huge, approximation is acceptable, and pre-computation is necessary
Distance Oracle

• A pre-computed data structure that enables us to find the (approximate) shortest distance between any two vertices in constant time.

• Oracle construction
  – Space complexity
    • linear, or sub-linear
  – Time complexity.
    • Quadratic complexity is unaffordable

• Query answering
  – Time complexity
    • constant time.
  – Quality
    • Approaching exact shortest distances
Current Solutions

• Theoretic distance oracle with performance bound:
  – Thorup-and-Zwick’s distance oracle [Thorup01]
  – Reduce the construction time on weighted [Baswana06] or unweighted graphs [Baswana062],
  – Reduce space cost on power-law graphs [Chen09] or ER random graphs [Enachescu08].

• Practical distance oracle: heuristic approaches
  – Sketch based [Potamias09, Gubichev10, Sarma10, Goldberg05, Tretyakov11]
  – Coordinate based [zhao2010,zhao2011]
Thorup-and-Zwick’s distance oracle

- $O(kn^{1+1/k})$ space, can be constructed within $O(kmn^{1/k})$ time
- Distance query can be answered in $O(k)$ time with at most $2k - 1$ multiplicative distance estimation.

- When $k = 1$, the distance oracle returns the exact distance but occupies quadratic space
- When $k = 2$, the worst distance is 3-times of the exact distance and the oracle occupies $O(n^{1.5})$ space
Coordinate-based solution

- **Idea**
  - Mapping all vertices into a hyperspace
  - Use the distance of two vertices in the new space to approximate the shortest distance in the graph

- **Steps**
  - Choose several landmarks (~100)
    - **Heuristics:** Degree, betweenness, ...
  - Precisely calculate the distance from each landmark to all other vertices by **BFS starting from each landmark**
  - Calculate the coordinates of landmarks by **simplex downhill** according to the precise distance among landmarks
  - Calculate the coordinates of other vertexes by **simplex downhill** according to the distance from these vertex to each landmark

- **Advantage**
  - Constant query time
  - Linear space
  - Linear construction time

Reference: [zhao 2010, zhao2011]
How to learn the coordinate

• Main method: Simplex downhill

• Objective function
  – Minimize
    \[ \sum_{u,v \in \text{Landmark}} (\text{CoordinateDist}(c(u), c(v)) - \text{RealDist}(u, v))^2 \]

• Space can be used
  – Euclidean space [zhao 2010]
  – Hyperboloid model [zhao 2011]

\[ \delta(x, y) = \arccosh \left( \sqrt{(1 + \sum_{i=1}^{n} x_i^2)(1 + \sum_{i=1}^{n} y_i^2) - \sum_{i=1}^{n} x_i y_i} \right) \cdot |c| \]

• Coordinate Dimension: \(~10\)
Sketch-based Solution

• Basic idea
  – Create a sketch of bounded size for each vertex
  – Estimate the distance using the sketch

• Advantage
  – linear space
  – If the sketch encodes enough useful information, it can produce highly accurate answers in short time (in most cases in constant time).

• Procedure
  – select landmark set \( L \)
  – Generate the shortest distances from each landmark to any other vertex
    • Sketch\([u]\) =\{(w_0, \delta_0), \ldots, (w_r, \delta_r)\}, where \( w_i \) is a vertex (called seeds) and \( \delta_i \) is the shortest distance between \( u \) and \( w_i \)
    – Estimate the distance between vertices \( u \) and \( v \) by the minimal value of \( d(u, w) + d(w, v) \) over all \( w \in L \).
Improvement on sketch-based solution

• Improve distance estimation
  – cycle elimination and tree-structured sketch [Gubichev10]
  – uses the distance to the least common ancestor of $u$ and $v$ in a shortest path tree [Tretyakov11]

• Improve landmark selection
  – optimal landmark selection is NP-hard
  – betweenness is a good landmark [Potamias09]
  – randomized seed selection [Sarma10]
Distance oracle summary

- Accuracy may be compromised
  - Node degrees, instead of their centrality, is usually used as a criterion for landmark selection to reduce the cost.

- Some distance oracles take very long time to create.
  - For example, in [Tretyakov11], it takes 23h to approximate betweenness for a 0.5 billion node graphs [Tretyakov11] even on a 32-core server with 256G memory.

- None of the previous distance oracles is designed for distributed graphs.
- Can hardly scale to billion node graphs on a general purpose graph system
Scale to billion node graphs with high accuracy

- Smart landmark selection
  - in local graphs

- Smart distributed BFS

- Smart answer generation rule

For details:
Towards a Distance Oracle for Billion Node Graphs, Microsoft Technical Report, 2012
Distributed BFS
Asynchronized BFS

- Asynchronized BFS (Bellman-ford)
  - In each iteration, update each vertex’s distance as the minimal distance of its neighbors plus one
  - Any time when a vertex distance is updated, it will trigger all its neighbors to update their distance
  - Until no distance update

- \(O(|V||E|)\), but flexible, allows fine-grained parallelism
Level-synchronized BFS [Andy05]

• Explore from a node level by level

• Iterate:
  – Each vertex of level $x$ sends distance update messages to their neighbors
  – Each vertex waits for messages for itself. If its distance is still unknown, update its distance as $x+1$
  – Synchronize at the end of each level

• $O(E)$ complexity
Possible Optimizations

• Observation:
  – Vertices of large degree are frequently visited even their distances to the source have already been computed.

• Optimization:
  – Cache the distances of large degree nodes on each machine

• “80/20” rule in real graphs

• Results on Trinity
  – 50% savings

• Scale-free graph
  \[ d(v) = \frac{1}{N^R} r(v)^R \]

For detail: Towards a billion-node graph distance oracle, Microsoft technique report, 2012
Betweenness computation
Betweenness computation

- Betweenness counts the fraction of shortest paths passing through a vertex

- Applications of betweenness
  - Vertex importance ranking
  - Landmark selection in distance oracle
  - Community detection

\[ C_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma_{st}(u)}{\sigma_{st}} \]

- Where, \( \sigma_{st} \) denotes the number of shortest paths from \( s \) to \( t \), \( \sigma_{st}(u) \) denote the number of shortest paths from \( s \) to \( t \) that pass through \( u \)
Betweenness computation on large graphs

- Exact betweenness costs $O(nm)$ time, unacceptable for large graphs [Brandes01]

- Lightweight approximate betweenness
  - Count shortest paths rooted at sampled vertices [Ercsey10]
  - Count shortest path with limited depth [Bader06][Brandes07]

- Parallelized exact betweenness [Bader06, Madduri09, Edmonds10, Tan09, Tu09]
  - On massive multithreaded computing platform [Bader06, Madduri09],
  - On distributed memory system [Edmonds10]
  - On multi-core systems [Tan09, Tu09].
Key algorithms in managing big graphs
A lightweight Distributed betweenness approximation

• On distributed platform
  – Compute the shortest paths within each machine
  – Local shortest path with two ends both in the local graph has high quality
  – Sample the local shortest with the probability proportional to its quality
  – Use the high-quality local shortest path to approximate exact betweenness

For detail: Towards a billion-node graph distance oracle, Microsoft technique report, 2012
Summary of key algorithms

• Most existing graph solutions are designed for centralized platforms
• Power of distributed computing has not been fully exploited for solving challenging graph problems
• Properties of distributed graphs have not been fully explored yet
• Previous existing solutions can hardly scale to real large graphs, especially billion node graphs
References

References-2

- [andy05] Andy Yoo, et al, A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L, SC’05
- [Pellegrini96] F. Pellegrini et al., “Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs,” in HPCN Europe ’96
- [Pregel] G. Malewicz et al., “Pregel: a system for large-scale graph processing,” in SIGMOD ’10

Key algorithms in managing big graphs
References-3

• [Bader06] D. A. Bader and K. Madduri, “Parallel algorithms for evaluating centrality indices in real-world networks,” in *ICPP ’06*
• [tu09] D. Tu and G. Tan, “Characterizing betweenness centrality algorithm on multi-core architectures,” in *ISPA’09*.
• [MapReduce] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,”
• [OSSDI, 2004.]

Key algorithms in managing big graphs
Thanks!
Who is GDM@FUDAN

- T-Robot
- Chinese Knowledge Graph
- Big graph management system
Framework

Web Resource

Entity/concept extraction
Entity Evaluation

Knowledge Extraction

Cloud based Graph DB engine

Applications
Reading
Search
QA

Conceptualization
Ssense Disambiguation

Knowledge base operations

Entity resolution
Relation Extraction

Entity Evaluation

Key algorithms in managing big graphs
中文知识图谱

• “云时代的四库全书”
• “Root of china”中国之根
• 链接: http://gdm.fudan.edu.cn/Soso/index.jsp
Thanks for your attentions!!

QA