Parallel Clustered Low-Rank Approximation and Its Application to Link Prediction*

Joyce Jiyoung Whang,
Department of Computer Science & Engineering, SKKU

* Published in *International Workshop on Languages and Compilers for Parallel Computing (LCPC)*, 2012.
Social Network Analysis

- Huge size of social network graphs poses great challenge on the analysis

  - Parallelization
    - Large scale distributed parallelization
  - Approximation
    - In many cases, approximate answers are sufficient, e.g. friend recommendations

- Two important ways to solve the challenge
- Over 900 million active users
- Over 500 million active users
- Over 175 million registered users
Need for Approximation

• Problem: compute the number of $\text{length-k}$ paths between every two vertices in a graph

• Solution 1: graph traversals
  – Too expensive for large graphs

• Solution 2: linear algebra formulation
  – Represent a graph by its adjacency matrix $A$
  – $A^K(i,j)$ is number of $\text{length-k}$ paths between vertices $i$ and $j$
  – Still very expensive
Low-rank Approximation

• The adjacency matrix of an undirected graph can be approximated by the product of three matrices $\mathbf{V}$, $\mathbf{D}$ and $\mathbf{V}^T$

\[ \mathbf{A} \approx \mathbf{V} \mathbf{D} \mathbf{V}^T \]

- $r \ll m$, called **rank**, is an input parameter
- The larger the $r$, the smaller the approximation error
- $\mathbf{V}^T \mathbf{V} = \mathbf{I}$ (Identity matrix)
Approximating $A^k$ by low-rank approximation.

$A_{mxm} \approx V_{mxr} D_{rxr} V_{mxr}^T$,

$A_{mxm} \approx V_{mxr} D_{rxr} V_{mxr}^T$,

$A_{mxm} \approx V_{mxr} D_{rxr} V_{mxr}^T$,

Large matrix multiplication becomes small matrix multiplication.
The Limitation of Low-rank Approximation

- Large rank $r$ is needed for large graphs to make the approximation error acceptable.
- The computation and memory costs are expensive for large graphs and rank.

How to improve it
Structure In Social Networks

- Not uniformly random graphs
- Clusters with few inter-cluster edges
- *Clustered* low-rank approximation
  1. Find clusters by partitioning
  2. Use low-rank approximation for each cluster
  3. Account for inter-cluster edges
Our Contributions

• A new parallel partitioning algorithm for social networks
  – Easy to parallelize
  – Compared to ParMetis
    • Faster and scales better
    • Generates similar quality partitions when ParMetis succeeds

• First parallel implementation of clustered low-rank approximation

• Application to link prediction of very large graphs
Matrix View of Clustering

\[ \begin{align*}
  n_1 & \quad n_2 & \quad n_3 & \quad n_4 \\
  n_1 & \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} & \quad n_2 & \quad n_3 & \quad n_4 \\
  n_1 & \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} & \quad 9
\end{align*} \]
Example: arXiv Network

21,363 vertices and 91,314 edges
Clustered Low-rank Approximation

Step 1: Graph Partitioning

<table>
<thead>
<tr>
<th>A_{11}</th>
<th>A_{12}</th>
<th>A_{13}</th>
<th>A_{14}</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_{31}</td>
<td>A_{22}</td>
<td>A_{23}</td>
<td>A_{24}</td>
</tr>
<tr>
<td>A_{31}</td>
<td>A_{32}</td>
<td>A_{33}</td>
<td>A_{34}</td>
</tr>
<tr>
<td>A_{41}</td>
<td>A_{42}</td>
<td>A_{43}</td>
<td>A_{44}</td>
</tr>
</tbody>
</table>

Step 2: Low-rank Approximation on diagonal blocks

\[ \mathcal{A}_{ii} \approx \mathcal{V}_i \mathcal{D}_{ii} \mathcal{V}_i^T \]

Step 3: Approximating Off-Diagonal blocks

\[ \mathcal{D}_{ij} \approx \mathcal{V}_i^T \mathcal{A}_{ij} \mathcal{V}_j \]

Compared to low-rank approximation
- Local structure to speed up computation
- Same storage of eigenvectors but higher rank
1. PEK: A new graph partitioning algorithm for social networks

- Intuition: High degree vertices capture the high level structure of such graphs
- PEK Algorithm:
  - Extract a small representative sub-graph (high degree vertices and their edges)
  - Partition this sub-graph
  - Propagate partitioning to entire graph
  - Refine with weighted kernel K-Means
Extract a Representative Sub-Graph

- Extract a small number of high-degree vertices and the edges between them
  - Graph is randomly and evenly distributed across processes
  - Each process selects its local vertices with degree larger than a threshold
  - Those vertices and the edges between them form the representative sub-graph
Partition Sub-graph

• Use ParMetis to partition sub-graph
  – Takes a small fraction of time

• Project partitions of vertices in sub-graph to original graph
  – Projected vertices assigned to partitions
  – Un-projected vertices are not assigned
Propagate Partitioning(1)

• Each partition has a virtual center point (centroid)
  – Initially computed based on the partitions of projected vertices

• Distance between a vertex to the centroid of a partition
  • Measure how close a vertex to the partition
  • Computed based on the partition size, #edges of the vertex to the partition, #edges within the partitions, etc.
Propagate Partitioning(2)

• Visit un-projected vertices in breadth-first order
  – Start from projected vertices
• For each un-projected vertex:
  – Assign to partition with the closest centroid
  – Update the centroid
• Each process has its own copy of all centroids
  – Do not synchronize updates of centroids
  – No impact on partition quality

Centroid of $Part_1$

Centroid of $Part_2$
Refine Partitions

• Iteratively improve initial partitioning
• On every iteration, each process:
  – Visits its local vertices **on the partition boundary**
  – For each boundary vertex \( v \):
    • Moves \( v \) from partition \( \text{Part}_i \) to \( \text{Part}_j \) if \( v \) is closer to \( \text{Part}_j \)
    • If moved, update the old and new centroids
• Processes synchronize updates of centroids once every iteration
  – Less communication
  – Does not degrade quality
2. Approximating Diagonal Blocks

- Assigns partitions to processes
- Each process computes low rank approximation on partitions independently

Sorted by the weights: \#nonzero(A_{ii}) x rank

Graph is reorganized after assignment

Assigns partition to the process currently having the least weights
3. Approximating Off-Diagonal Blocks

- Undirected graph => symmetric adjacency matrix $A$
  - Only one of $A_{ij}$ and $A_{ji}$ needs to be approximated
- A job, $J_{i,j}$, $i < j$, denotes approximating either $A_{ij}$ or $A_{ji}$
  - Private jobs of process $P_i$, e.g. $J_{1,4}$
    - Can be finished by $P_i$ without communication
  - Shared jobs between $P_i$ and $P_j$, e.g. $J_{2,3}$
    - Either $P_i$ or $P_j$ can finish it
    - Communication is needed between $P_i$ and $P_j$
- Processes first finish its private jobs
- Dynamic load balancing for scheduling shared jobs
Experimental Setting

• Machine: Ranger (Texas Advanced Computing Center)
  – Each node has a 4 x 4-core AMD Opteron 2.2GHz CPU and 32GB memory.
  – InfiniBand networks with 5GB/s point-to-point bandwidth
• Libraries: Intel ICC 10.1, OpenMPI 1.3, ARPACK++, GotoBLAS 1.3 and Elemental 1.7
• Assign one process per node
Datasets

- Converted the graphs to undirected graphs, the table shows the statistics of graphs after conversion

<table>
<thead>
<tr>
<th>Name</th>
<th>#Vertices</th>
<th>#Edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocLive</td>
<td>3,828,682</td>
<td>39,870,459</td>
<td>LiveJournal online social network</td>
</tr>
<tr>
<td>Twitter_10M</td>
<td>11,316,799</td>
<td>63,555,738</td>
<td>Twitter social network</td>
</tr>
<tr>
<td>Twitter_40M</td>
<td>41,652,230</td>
<td>1,202,513,046</td>
<td>Twitter social network</td>
</tr>
</tbody>
</table>
Runtime and Speedup of Parallel Clustered Low-rank Approximation

- **#Partitions**
  - SocLive: 500
  - Twitter_10M: 500
  - Twitter_40M: 1000

- **Rank for Diagonal Phase**
  - SocLive: 100
  - Twitter_10M: 100
  - Twitter_40M: 100
Runtime and Speedup in Each Phase

- Partitioning and offDiagonal phases scale well
Load Balancing of diagonal and offDiagonal Phases

- #Partitions is small compared to #Processes, not enough space for load balancing in diagonal phase

#Partitions:
- SocLive: 500
- Twitter_10M: 500
- Twitter_40M: 1000
Graph Partitioning
Comparing PEK with ParMetis

- #Partitions: 500
- Degree Threshold:
  - SocLive: 42 (5% vertices)
  - Twitter_10M: 200 (less than 5% vertices)
- Cut-size and NormCut
  - Lower is Better
  - cut-size: the edges across partitions
  - NormCut: normalized cut-size by the total degree of vertices of each partition
  - divided by the number of clusters
- ParMetis cannot partition Twitter_40M since the memory is not enough
Link Prediction

• Our parallel clustered low-rank approximation enabled first ever study of Katz measure on large real-world social networks

\[ \text{Katz}(v_i, v_j) = \sum_{k=1}^{\infty} k \left| \text{paths}_{v_i \rightarrow v_j}^{\text{length}=k} \right| \text{ where } \beta \text{ is damping factor} \]

• Randomly remove 30% edges from graphs and perform link prediction on the resulting graphs.

• Precision is the ratio of correct predictions in top-\(k\) predictions

#partitions:500
rank of diagonal:50

#partitions:500
Rank of diagonal:100
Conclusion

• Developed a new graph partitioning algorithm for social networks
  – Fast and scales well to large number of processes
  – Faster than ParMetis and similar partition quality as ParMetis

• Parallelized clustered low-rank approximation and applied it on large real-world social networks

• Benchmark combines:
  – Irregular and regular computations
  – Dense and sparse data structures

• Approximation and Parallelization are the keys for solving large-scale social network problems