Fast Algorithm for Modularity-based Graph Clustering

Hiroaki Shiokawa
NTT Software Innovation Center, NTT Corporation,
July 23rd, 2013
BACKGROUND & MOTIVATION
Large Graphs

- Large-scale graphs become available
  - Facebook: 1.11 billion active users / month (*1)
  - Twitter: 140 million active users / day
    340 million new posts / day (*2)
  - And more …

- A lot of techniques for analyzing massive-scale graph
  - Massive data require so much time for analysis 😞
  - It is important to analyze large scale data quickly 😊

(*1) “Key Facts”, http://newsroom.fb.com/Key-Facts
(*2) http://dev.twitter.com/media/authors
Graph Clustering

- **Graph clustering is one of the most important methods**
  - Community detection over social networks
  - Event detection from microblogging services
  - Brain Analysis, Image segmentation, …

![Graph Clustering Diagram]
Modularity-based Graph Clustering

• Clustering methods which find the division of graph to maximize the modularity measure
• Improvement of clustering speed

Our research target

There are no algorithms 😞

1B ~ 100M nodes/hour

10M nodes/hour

1M nodes/hour

100k nodes/hour

10k nodes/hour

BGLL [Blondel et al., 2008]

CNM [Clauset et al., 2004], WT [Wakita et al., 2008]

Newman method [Newman et al., 2004]

Girvan-Newman method [Girvan et al., 2004]
Objective and Contributions

• Objective

Fast graph clustering method with high modularity

• 3 key techniques
  1. Incremental nodes aggregation
  2. Incremental nodes pruning
  3. Efficient ordering of nodes selections

• Contributions of our algorithm

  • Efficiency
    • Considerably faster than BGLL
    • Clusters 100M nodes within 3 minutes
  • High Modularity
    • Scores high modularity as same as BGLL
  • Effectiveness
    • Improves performances for complex networks
PRELIMINARIES
Modularity

- **Modularity evaluates the strength of division of a graph into clusters** [Newman and Girvan 2004]
  - Finding the division which maximizes modularity is NP-complete
  \[ \Rightarrow \text{A lot of greedy approaches were proposed} \]

\[ Q = \sum_{i \in C} \left\{ \frac{e_{ii}}{2m} - \left( \frac{\sum_{j \in C} e_{ij}}{2m} \right)^2 \right\} \]

- \( C \) : Set of cluster
- \( e_{ij} \) : Number of edges between cluster \( i, j \)
- \( m \) : Total number of edges in a graph

The fraction of the edges within cluster \( i \)
The fraction of outgoing edges from a cluster \( i \)
State of the art algorithm: BGLL

- Continuing following passes until the modularity score is maximized

Pass

1\textsuperscript{st} phase: Local clustering
1) Selects a node
2) Computes the modularity gain
3) Places the neighbor node in the same cluster

2\textsuperscript{nd} phase: Data reduction
• Aggregates all nodes in the same cluster as a single node

Continue to the next pass
PROPOSED ALGORITHM
Overview of proposed algorithm

Our method

• **3 key techniques**
  • Incremental aggregation
  • Incremental pruning
  • Efficient ordering

BGLL

• Batch based aggregation
• Random ordering

• Clustering coefficient
• Power-law degree distribution
Idea 1: Clustering coefficient

- **Complex networks have large clustering coefficient**
  - Clustering coefficient is a measure of degree to which nodes in a graph tend to cluster together
  - **There are many duplicated nodes/edges** in a graph which has large clustering coefficient

![Diagram showing internal nodes within a cluster, duplicated edges between different clusters, and nodes whose clusters are trivially obtained.](image-url)
Idea 2: Power-law degree distribution

- Complex networks have highly skewed degree distribution following the power-law distribution
  - Most of nodes only have a few neighbor nodes, and only few nodes have large number of neighbor nodes
  - The frequency $F$ of nodes with degree $d$ is $F \propto d^{-\alpha}$

Example of degree distribution of complex network

- Random ordering of node selection leads redundant computation
Incremental Aggregation

- Incrementally aggregate nodes which belong to the same cluster
  - It aggregates duplicated edges between clusters while keeping the graph semantics
  - Example

![Diagram showing incremental aggregation](image)
Incremental pruning

• **Incrementally prune nodes whose cluster is trivially obtained**
  • We can easily assign nodes to clusters without computing modularity gains
  • From the graph structure, there are 2 patterns of pruning

**Pattern A**
A node only has a single neighbor node

**Pattern B**
A node surrounded by nodes belong to same cluster

Easy to prune nodes 😊

Non-trivial 😞

Check neighbors’ cluster
Incremental pruning (Cont.)

- **Efficient pruning approach for pattern 2**
  - All nodes within the same cluster have been aggregated to a node by incremental aggregation
  - *We can find all prunable nodes by obtaining nodes such that they have only a single adjacent node*
Efficient ordering of node selection

- Dynamically selects a node with the smallest degree
  - *Example*) Node A and B being assigned to the same cluster

By selecting node with the smallest degree, we can avoid producing super-cluster structures.

It’s more efficient to compute from node B than node A

Select from A
many computations
Select from B
2 computations
EVALUATION
Datasets & Experimental Environment

- **Real world datasets**
  - 2 Social networks and 3 Web graphs of IP domains

| Dataset      | $|V|$     | $|E|$   | Skewness of degree distribution $\alpha$ |
|--------------|---------|--------|----------------------------------------|
| dblp-2010    | 326,186 | 1,615,400 | 2.82                                  |
| ljourn-2008  | 5,363,260 | 79,023,142 | 2.29                                  |
| uk-2005      | 39,459,925 | 936,364,282 | 1.71                                  |
| webbase      | 118,142,155 | 1,019,903,190 | 2.14                                  |
| uk-2007-05   | 105,896,555 | 3,738,733,648 | 1.51                                  |

- **Experimental Environment**
  - All experiments were conducted on a Linux 2.6.18 server with Intel Xeon CPU L5640 2.27GHz and 144GB RAM
  - Run all methods on 1 core, 1CPU
Computational time

- Proposed is up to **60 times faster** than the state of the art algorithm BGLL
- Graphs with highly skewed degree distribution run faster than the other datasets

- 100 million nodes and 1 billion edges in 156 seconds!
- 320k nodes within 1 seconds!
Computational time – power-law differences

Figure 2: Power-law difference
Computational time – size differences

Figure 3: Scalability
Modularity score

- **Modularity score for datasets**
  - Large modularity score means the output of algorithms is well clustered
  - Proposed method achieves *almost same modularity scores as/slightly higher than BGLL*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td>0.90</td>
<td>0.74</td>
<td>0.98</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>BGLL</td>
<td>0.88</td>
<td>0.74</td>
<td>0.97</td>
<td>0.96</td>
<td>0.97</td>
</tr>
</tbody>
</table>
CONCLUSION
Conclusion

• Fast clustering algorithm for large graphs
  • Our solution
    • Incremental aggregation
    • Incremental pruning
    • Efficient ordering of nodes selections

• Contribution of our algorithm
  • Efficiency
    • Considerably faster than BGLL
    • Clusters 100M nodes within 3 minutes
  • High Modularity
    • Scores high modularity as same as BGLL
  • Effectiveness
    • Improves performance for complex networks