Fast Algorithms for Community Detection

Kousuke Tachida
Frontiers of Network Science Fall 2018
Fast Algorithms for Community Detection

1. Louvain Algorithm
2. Infomap

- Accuracy is comparable to other community detection algorithms
- Better scalability!
- Not guaranteed to be the fastest or most accurate
Motivation

• Greedy algorithm for community detection is $O(N^2)$
• Bad performance for very large networks
Algorithm Similarities

• Both optimize quality function $Q$
  • Louvain: $Q$ is modularity
  • Infomap: $Q$ is entropy-based measure called the map equation

• Use same optimization procedure
Louvain Algorithm

• “Fast unfolding of communities in large networks” (2008)
• Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre

• Optimize modularity: [-1, 1], the density of links inside communities compared to links between communities

\[ M = \frac{1}{2W} \sum_{ij} \left[ w_{ij} \left( 1 - \frac{k_i k_j}{2W} \right) \delta(c_i, c_j) \right] \]

- \( w_{ij} \) is the edge weight between nodes i and j
- \( k_i \) is the sum of the weights of links that connect to node i
- \( W \) is the sum of the weights of all links in the network
- \( c_i \) is the community of node i
- \( \delta \) is a simple delta function (1 if \( c_i = c_j \) else 0)
Louvain Step 1: Local Modularity Optimization

• Start with a weighted network of N nodes, initially assigning each node to a different community

• For each node i:
  • Evaluate change in modularity if we placed node i in the community of each of its neighbors
    \[ \Delta M = \left[ \frac{\Sigma_{in} + 2k_{i,in}}{2W} - \left( \frac{\Sigma_{tot} + k_i}{2W} \right)^2 \right] - \left[ \frac{\Sigma_{in}}{2W} - \left( \frac{\Sigma_{tot}}{2W} \right)^2 - \left( \frac{k_i}{2W} \right)^2 \right] \]
    • \( \Sigma_{in} \) is the sum of the weights of links inside community C
    • \( \Sigma_{tot} \) is the sum of the weights of links to all nodes in C
    • \( k_i \) is the sum of the weights of links that connect to node i
    • \( k_{i,in} \) is the sum of the weights of links from node i to nodes in C
    • W is the sum of the weights of all links in the network
  • Move node i to the community with biggest modularity gain
  • If no positive gain is found, stay in original community

• Repeat until no further improvement can be achieved
Louvain Step 2: Community Aggregation

• Construct a new network whose nodes are the communities found in step 1
• Weight of a link between two nodes is the sum of the weights of all the links between the nodes in the corresponding communities
  • Links between nodes of the same community lead to weighted self-loops
• Once step 2 is completed, a single pass of the algorithm is completed
• Repeat steps 1 and 2, decreasing the number of communities with each pass, until maximum modularity is achieved
Louvain Algorithm: Illustration
Louvain Algorithm: Complexity and Results

• More limited by storage demands than computational time
• $O(L \log L)$ or $O(N \log N)$ for sparse networks
  • First pass requires most computational effort
  • Subsequent passes are over decreasing amount of nodes and links

<table>
<thead>
<tr>
<th></th>
<th>Karate</th>
<th>Arxiv</th>
<th>Internet</th>
<th>Web nd.edu</th>
<th>Phone</th>
<th>Web uk-2005</th>
<th>Web WebBase 2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes/links</td>
<td>34/77</td>
<td>9k/24k</td>
<td>70k/351k</td>
<td>325k/1M</td>
<td>2.6M/6.3M</td>
<td>39M/783M</td>
<td>118M/1B</td>
</tr>
<tr>
<td>Clauset, Newman,</td>
<td>.38/0s</td>
<td>.772/3.6s</td>
<td>.692/799s</td>
<td>.927/5034s</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
</tr>
<tr>
<td>and Moore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pons and Latapy</td>
<td>.42/0s</td>
<td>.757/3.3s</td>
<td>.729/575s</td>
<td>.895/6666s</td>
<td>-/-</td>
<td>-/-</td>
<td>-/-</td>
</tr>
<tr>
<td>Watiye and Tsurui</td>
<td>.42/0s</td>
<td>.781/0.7s</td>
<td>.667/62s</td>
<td>.898/248s</td>
<td>.56/464s</td>
<td>-/-</td>
<td>-/-</td>
</tr>
<tr>
<td>Louvain Method</td>
<td>.42/0s</td>
<td>.813/0s</td>
<td>.781/1s</td>
<td>.935/3s</td>
<td>.769/134s</td>
<td>.979/738s</td>
<td>.984/152mn</td>
</tr>
</tbody>
</table>

• -/- means that the method took over 24 hours to run
Louvain Algorithm: Problems

• Resolution limit: hard to detect smaller communities, may be merged into larger communities
  • Intermediate steps may have more intuitive structures

• Degeneracy problem: exponentially large number of possible community assignments with close to maximum modularity
  • Hard to find global maximum (NP-hard)
  • Is the global maximum better, more scientifically important than other community assignments with similar modularity?
  • Locally optimal community assignments can have different structural properties
Infomap

- Martin Rosvall and Carl T. Bergstrom
- Exploits data compression, compressing the movement of a random walker on a network
  - Finding structure in networks is equivalent to solving a coding problem
- Optimizes a quality function for networks known as the map equation
Infomap: Goal

• If we want to understand how network structure relates to system behavior, we need to understand the flow of information on the network

• A group of nodes among which information flows quickly and easily can be aggregated as a single well-connected module/community

• Succinctly describing information flow is a coding/compression problem

• Use a random walk as a proxy for information flow
  • Random walk uses all of the information in the network representation and nothing more

• Finding community structure in networks is equivalent to solving a coding problem
  • Want a compressed description of a random walk, with unique names for important structures
Infomap: The Example

• Weighted network with $n = 25$ nodes
• 71-step realization of a random walk
• Compress the description of the random walk AND give important structures unique names
Attempt 1

• Give a unique name to each node
• Derived using a Huffman encoding, using the estimated probability that the random walk visits that node
  • Huffman encoding assigns shorter names to more common nodes
• The random walk can be represented in 314 bits
Attempt 2

- Use a two level description, retaining unique names for each module, but use a different Huffman code for nodes within each cluster
  - Analogy: US cities and street names
- Start with name of the cluster, then the node names within the cluster
- Exit code is chosen as part of the within-cluster Huffman coding, indicating the walk is leaving the current cluster
  - Exit code is followed by the name of the next cluster
- Yields on average a 32% shorter description
- The random walk can be represented in 243 bits
Illustration

- Fine-grain (index codebook) and coarse-grain (module codebook) representations
Infomap: Map Equation

• Look for a module partition $M$ of $n$ nodes into $m$ modules to minimize the expected description length of a random walk

• Average description length is given by the map equation or $L$

• $L(M) = q \cdot H(Q) + \sum_{i=1}^{m} p_{i} H(P_{i})$
  • First term: Average number of bits necessary to describe movement between communities
  • Second term: Average number of bits necessary to describe movement within communities
    • Exiting is also considered a movement
Infomap: Map Equation

\[ L(M) = q_i \cdot H(Q) + \sum_{i=1}^{m} p_i \cdot H(P_i) \]

- Per-step probability that the random walker switches modules
- \( q_i \sim \) is the per-step probability that the walker leaves module \( i \)

\[ H(Q) = \sum_{i=1}^{m} \frac{q_i \sim}{\sum_{j=1}^{m} q_j \sim} \log \left( \frac{q_i \sim}{\sum_{j=1}^{m} q_j \sim} \right) \]
- \( H(Q) \) is the entropy of movements between modules

\[ p_i \sim = q_i \sim + \sum_{\alpha \in i} p_{\alpha} \]
- Weights the entropy of movements within module \( i \)
- \( p_{\alpha} \) is the ergodic node visit frequency at node \( \alpha \) within the random walk

\[ H(P_i) = \frac{q_i \sim}{q_i \sim + \sum_{\beta \in i} p_{\beta}} \log \left( \frac{q_i \sim}{q_i \sim + \sum_{\beta \in i} p_{\beta}} \right) \]
\[ + \sum_{\alpha \in i} \frac{p_{\alpha}}{q_i \sim + \sum_{\beta \in i} p_{\beta}} \log \left( \frac{p_{\alpha}}{q_i \sim + \sum_{\beta \in i} p_{\beta}} \right) \]
- \( H(P_i) \) is the entropy of movements within module \( i \)
Minimizing the Map Equation

• Use the same optimization procedure as in the Louvain algorithm
• Start with each node in a different community
  • Move nodes to neighboring communities for greatest decrease in L
  • Aggregate communities
  • Repeat until L is minimized
Infomap: Complexity

• Computational complexity is determined by the procedure used to minimize the map equation

• So it becomes the same as the Louvain algorithm
  • $O(L \log L)$ or $O(N \log N)$ for a sparse network
Conclusion

• Louvain and Infomap algorithms offer tools for fast community identification

• Comparable accuracy to other algorithms
Sources

• http://networksciencebook.com/chapter/9#advanced-9c
• https://arxiv.org/abs/0803.0476
• https://en.wikipedia.org/wiki/Louvain_Modularity
• https://perso.uclouvain.be/vincent.blondel/research/louvain.html
• https://neo4j.com/docs/graph-algorithms/current/algorithms/louvain/
• https://arxiv.org/abs/0707.0609
• http://www.tp.umu.se/~rosvall/livemod/mapequation/