Layered Label Propagation: A Coordinate-Free Node Ordering Method for Graph Compression

Abstract

We present a summary of how Layered Label Propagation (LLP), a highly scalable, coordinate-free, **graph reordering algorithm**, uses community finding techniques to permute very large immutable graphs, with applications to **graph compression**.

Motivation/Problem

Large graphs (millions or billions of nodes) may not fit in main memory.

- Standard graph mining algorithms usually assume they do.

What it is

A graph node ordering algorithm.

- Not a compression algorithm.

Why do we need it?

Most current algorithms are sensitive to the initial ordering of the graphs.

- Different compression ratios depending on how the dataset is originally presented.
- Layered Label Propagation (LLP) is **efficient** and **coordinate-free**.
  - Attains similar results independently of the initial ordering.

What does it try to acomplish?

- Effective **general** techniques to store and access graphs.
- Resulting compressed data structure must provide **fast amortised random access** to an edge.

General idea

- Combines with the Boldi and Vigna (BV) compression algorithm.
  - "*de facto* standard for handling large web-like graphs".
- Exploits the inner structure of the network to devise **intrinsic** orderings.
  - Approached as a **community finding** problem.

General idea

Layered Label Propagation

- **Coordinate-free**.
- Exploits **similarity** and **locality**.
  - **similarity**: nodes tend to have resembling sets of neighbours if they're close to each other in the ordering.
  - **locality**: most of the edges are shared between nodes close to each other in the ordering.

How does it work?

Standard Label Propagation

- One of a class of community finding algorithms.
- Fits this problem well because it is:
  - general (no *a priori* information is needed regarding the structure of the network).
  - efficient (requires only a few passes through the graph).

How does it work?

Label Propagation

- At the beginning each node is assigned a unique label.
- At each iteration, each node takes the label that most of its neighbours have.
    - Ties are resolved randomly.
- As the labels propagate, densely connected groups of nodes are formed.
- This iterative process goes on until every node in the graph is assigned a label equal to most of its neighbours.
- At the end, nodes with the same labels are grouped together as communities.

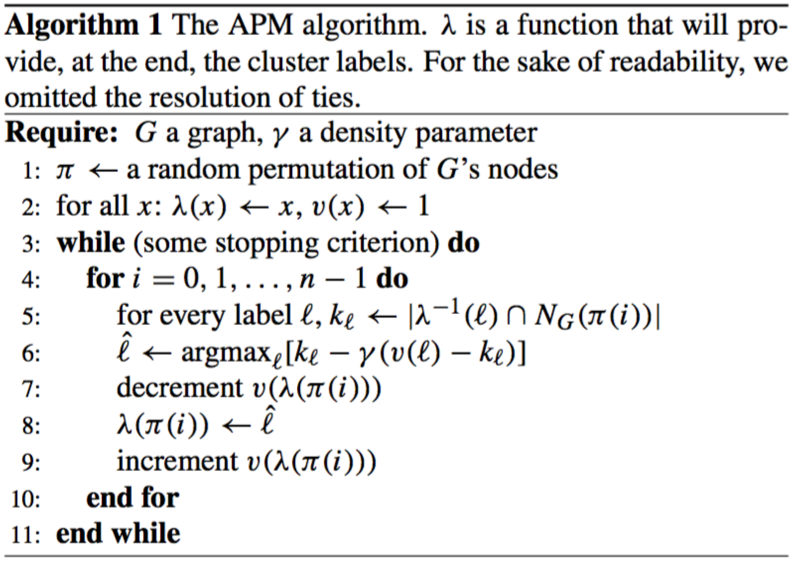
How does it work?

Absolute Potts Model (APM)

- Variant of label propagation, with new label update rule.
- Addresses the resolution limit problem in community detection by introducing a new weight parameter.
  - **Standard label propagation**: label chosen is the one that maximizes k_i (number of neighbours with label λ_i)
  - **APM**: maximize $k_i - \gamma(v_i - k_i)$
    - v_i is the number of nodes in the network with label λ_i.

How does it work?

Absolute Potts Model (APM)



Algorithm 1 The APM algorithm. λ is a function that will provide, at the end, the cluster labels. For the sake of readability, we omitted the resolution of ties.

```
Require: $G$ a graph, $\gamma$ a density parameter
 1: $\pi \leftarrow$ a random permutation of $G$'s nodes
 2: for all $x$: $\lambda(x) \leftarrow x$, $v(x) \leftarrow 1$
 3: while (some stopping criterion) do
 4:     for $i = 0, 1, \ldots, n-1$ do
 5:         for every label $\ell$, $k_\ell \leftarrow |\lambda^{-1}(\ell) \cap N_G(\pi(i))|$
 6:         $\hat{\ell} \leftarrow \operatorname{argmax}_\ell [k_\ell - \gamma(v(\ell) - k_\ell)]$
 7:         decrement $v(\lambda(\pi(i)))$
 8:         $\lambda(\pi(i)) \leftarrow \hat{\ell}$
 9:         increment $v(\lambda(\pi(i)))$
10:     end for
11: end while
```

How does it work?

Absolute Potts Model (APM)

Shortcomings

- no way to predetermine an “optimal” value for γ
- yields an enormous amount of huge-sized clusters.

How does it work?

LLP

- Starts with any initial ordering of the nodes.
- Applies APM iteratively, with different values of γ, computing a new ordering each time.
  - $x \leq_{k+1} y \begin{cases} \pi_k(\lambda_k(x)) < \pi_k(\lambda_k(y)) \\ \lambda_k(x) = \lambda_k(y) \wedge \pi_k(x) < \pi_k(y) \end{cases}$
- γ values are picked uniformly randomly from the set $\{0\} \cup \{2^{-i}, i = 0, ..., K\}$.

Why is it good?

Host transition

$$HT(\mathcal{H}, \pi) = 1 - \frac{\sum_{i=1}^{|V_G|-1} \delta(\mathcal{H}[\pi^{-1}(i)], \mathcal{H}[\pi^{-1}(i-1)])}{|V_G| - 1}$$

Variation of Information

$$VI(\mathcal{H}, \mathcal{H}_{|\pi}) = H(\mathcal{H}_{|\pi}) - H(\mathcal{H})$$

$$H(\mathcal{U}) = -\sum_{i=0}^{R} P(i)\, log(P(i)), \quad P(i) = \frac{|\mathcal{U}_i|}{|\mathcal{V}_i|}$$

Highly parallel

References

- P. Boldi, M. Rosa, M. Santini and S. Vigna. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. arXiv:1011.5425v2.
- U. Raghavan, R. Albert and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. arXiv:0709.2938v1.
- P. Ronhovde and Z. Nussinov. Local resolution-limit-free Potts model for community detection. arXiv:0803.2548v4.