Louvain Community Detection in Connectomes

BY MARK HORENI
The Problem

A. Monadic
B. Dyadic
C. Triadic

D. Image of neural connections with labels: VNC, VA4, rL,Amp
The Problem
The Data

- **C. elegans Connectome**
  - 279 Nodes
  - 3225 Edges
- **Mouse Retina**
  - 1123 Nodes
  - 577,350 Edges
- **Human Connectome**
  - MRI Data
  - 277,345 Nodes
  - 64.4M Edges
Modularity

- Metric to determine density of communities compared to null model
- Global Property
- Goal: Maximize Modularity
- Suffers from “Resolution Limit”

\[ Q = \frac{1}{2m} \sum_{ij} \left[ A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j), \]
Louvain Visualized
1: $V$: a set of vertices
2: $E$: a set of edges
3: $W$: a set of weights of edges, initialized to 1
4: $G \leftarrow (V, E, W)$
5: repeat
6: $C \leftarrow \{\{v_i\} | v_i \in G(V)\}$
7: calculate current modularity $Q_{\text{cur}}$
8: $Q_{\text{new}} \leftarrow Q_{\text{cur}}$
9: $Q_{\text{old}} \leftarrow Q_{\text{new}}$
10: repeat
11: for $v_i \in V$ do
12: $Q_{\text{cur}} \leftarrow Q_{\text{new}}$
13: remove $v_i$ from its current community
14: $N_{v_i} \leftarrow \{c_k | v_i \in G(V), v_j \in c_k, e_{ij} \in G(E)\}$
15: find $c_x \in N_{v_i}$ that has $\max \Delta Q_{\{v_i\}, c_x} > 0$
16: insert $v_i$ into $c_x$
17: end for
18: calculate new modularity $Q_{\text{new}}$
19: until no membership change or $Q_{\text{new}} = Q_{\text{cur}}$
20: $V' \leftarrow \{c_i | c_i \in C\}$
21: $E' \leftarrow \{e_{ij} | \forall e_{ij} \text{ if } v_i \in C_i, v_j \in C_j, and C_i \neq C_j\}$
22: $W' \leftarrow \{w_{ij} | \sum w_{ij}, \forall e_{ij} \text{ if } v_i \in C_i \text{ and } v_j \in C_j\}$
23: $G \leftarrow (V', E', W')$
24: until $Q_{\text{new}} = Q_{\text{old}}$
Enhanced Kernel

- Every process gets its own set of vertices
- Consists of 2 Parts
  - Iterations of Louvain Locally
    - Maximizes Modularity in local communities
    - Aggregates to find a global modularity
    - Uses Ghost Vertices for interprocess communication
  - Building a new Graph
    - Communicate with other processes to construct new communities
Building the graph
Enhanced Pseudocode

Algorithm 2: Parallel Louvain Algorithm (at rank $i$).
Input: Local portion $G_i = (V_i, E_i)$ of the graph $G = (V, E)$
Input: Threshold, $\tau$ (default: $10^{-6}$)

1: $C_{curr} \leftarrow \{u\} | \forall u \in V$
2: $\{currMod, prevMod\} \leftarrow 0$
3: while true do
4:     $currMod \leftarrow$ LouvainIteration($G_i, C_{curr}$)
5:     if $currMod - prevMod \leq \tau$ then
6:         break and output the final set of communities
7:     BuildNextPhaseGraph($G_i, C_{curr}$)
8:     $prevMod \leftarrow currMod$

Algorithm 3: Algorithm for the Louvain iterations of a phase at rank $i$.
Output: Modularity at the end of the phase.

function LOUVAINITERATION($G_i, C_{curr}$)
2: $V_i \leftarrow$ ExchangeGhostVertices($G_i$)
3: while true do
4:     send latest information on those local vertices that are stored as ghost vertices on remote processes
5:     receive latest information on all ghost vertices
6:     for $v \in V_i$ do
7:         Compute $\Delta Q$ that can be achieved by moving $v$ to each of its neighboring communities
8:     Determine target community for $v$ based on the migration that maximizes $\Delta Q$
9:     Update community information for both the source and destination communities of $v$
10:    send updated information on ghost communities to owner processes
11:    $C_{info} \leftarrow$ receive and update information on local communities
12:    $currMod, \tau$ Compute modularity based on $G_i$ and $C_{info}$
13:    $currMod \leftarrow$ all-reduce: $\sum_{v, currMod_v}$
14:    if $currMod - prevMod \leq \tau$ then
15:        break
16:    $prevMod \leftarrow currMod$
17: return $prevMod$
Enhanced Implementation (Grappolo)

- Public Library in C++
- ~500 Lines
- Uses MPI for interprocess Communication
- CSR to store Graphs
- Looked at time difference in “Resolution” Value
Results

Wall Clock Time

Number of Iterations
Compared to Sequential

Sequential

Parallel

Resolution 0.5 0.75 1.0

Resolution 0.5 0.75 1.0

Time

Time

10^2

10^1

10^0

10^1

10^0

10^-1

10^-2

279 1123 277345

279 1123 277345
What I’ve Learned

- Louvain is really fast, especially when parallelized
- Python not as fast
- Modularity may not mean anything in real life
- Will use it as a baseline since it’s so fast
Questions?