ARCHER

This is the official implementation of **ARCHER** (Adaptive RWR Computation on Hypergraphs), which is described in the following papaer:

- **Random Walk with Restart on Hypergraphs: Fast Algorithms and Applications**
  Jaewan Chun, Geon Lee, Kijung Shin and Jinhong Jung
  ECML PKDD 2023

Overview

Random walk with restart (RWR) is a widely-used measure of node similarity in graphs, and it has proved useful for ranking, community detection, link prediction, anomaly detection, etc. Extension of RWR to hypergraphs, however, has been unexplored despite its great potential. Since RWR is typically required to be computed separately for a larger number of seed nodes or even for all nodes, fast computation of it is indispensable.

In this paper, we propose **ARCHER**, a fast computation method for RWR on hypergraphs. Specifically, we first formally define RWR on hypergraphs, and then we propose two constituent computation methods of it. Since their relative performance heavily depends on datasets, we also develop a method for automatic selection between them that takes a very short time compared to the total running time. Through our extensive experiments on 18 real-world hypergraphs, we show that **ARCHER** is **(a) Fast & Space-Efficient:** requiring up to $137.6\times$ less preprocessing time, $218.8\times$ less query time, and $16.2\times$ less space than using always one constituent method, **(b) Intelligent:** selecting the better constituent method in almost all (spec., 17 out of 18) cases, and **(c) Versatile:** demonstrating successful applications to anomaly detection and similar node retrieval on hypergraphs.

Datasets and Settings

We provide code of ARCHER using BePI and BEAR as in the paper. We provide the information of the datasets used in the experiment below.

Dataset	Nodes	Hyperedges	$\text{avg}_{e\in E}\lvert e\rvert$	$\max_{e\in E}\lvert e\rvert$	Density	Overlapness
EEN	143	10,883	2.47	37	76.12	188.21
EEU	998	234,760	2.39	40	234.09	559.80
SB	294	29,157	7.96	99	99.17	789.62
HB	1,494	60,987	20.47	399	40.82	835.79
WAL	88,860	69,906	6.59	25	0.79	5.81
TRI	172,738	233,202	3.12	85	1.35	4.21
AM	55,700	105,655	8.12	555	1.90	15.41
YP	25,252	25,656	18.2	649	1.02	18.50
TW	81,305	70,097	25.2	1205	0.86	21.75
COH	1,014,734	1,812,511	1.32	925	1.75	2.32
COG	1,256,385	1,590,335	2.80	284	1.26	3.53
COD	1,924,991	3,700,067	2.79	280	1.92	5.35
THU	125,602	192,947	1.80	14	1.54	2.76
THM	176,445	719,792	2.24	21	4.08	9.13
THS	2,675,955	11,305,343	2.23	67	4.22	9.56
ML1	3,533	6,038	95.3	1435	1.71	162.83
ML10	10,472	69,816	84.3	3375	6.67	562.02
ML20	22,884	138,362	88.1	4168	6.05	532.93

For experiments of preprocessing and query costs, we set the restart probability to 0.05. And we set k in reordering method to 0.001 for BEAR and 0.2 for BePI. The error tolerance for BePI is set to 10^{-9}. This implementation has been written in MATLAB and tested with MARLAB R2021a. We conducted our experiments on a workstation with AMD Ryzen 9 3900X and 128GB memory.

Demo

We included two demo codes, ARCHER_BEAR_demo.m, ARCHER_BePI_demo.m which runs ARCHER on an example dataset each using BEAR and BePI as preprocessing methods. Description of the data is given in “Dataset/readme.pdf”.

How to use ARCHER in My Code

In src/ARCHER_BEAR and src/ARCHER_BePI, we provide functions for preprocessing and query of ARCHER each using BEAR and BePI.

To use ARCHER with BEAR, please add paths as follows:

```
addpath('src/ARCHER_BEAR');
addpath(genpath('src/BEAR'));
```

Then, we can use ARCHER using BEAR as follows:

```
[PREP,is_clique] = ARCHERPre_BEAR(R,W,c);
r = ARCHERQuery_BEAR(s, c, PREP, is_clique, num_v);
```

To use ARCHER with BePI, please add paths as follows:

```
addpath('src/ARCHER_BePI');
addpath(genpath('src/bepi'));
```

Then, we can use ARCHER using BePI as follows:

```
[PREP, is_clique] = ARCHERPre_BePI(R,W,c);
r = ARCHERQuery_BePI(s, c, epsilon, PREP, is_clique, num_v);
```

We describe the inputs of our functions on the following table.

Input	Description
R	node-weight matrix ($\|E\| \times \|V\|$) $R_{ev} = \gamma_e(v)$ if $v \in e$, and 0 otherwise ($\gamma_e(v)$: edge-dependent node weight)
W	hyperedge-weight matrix ($\|V\| \times \|E\|$) $W_{ve} = \omega(e)$ if $v \in e$, and 0 otherwise ($\omega(e)$: hyperedge weight)
c	restart probability of RWR
s	seed (or source) node
epsilon	error tolerance for BePI
PREP	preprocessed matrices
is_clique	if ARCHER selected clique expansion true, if ARCHER selected star expansion false
num_v	number of nodes