Our Goal: develop a scalable accurate method to compute random walk with restart (RWR), a widely-used measure of relevance between two nodes in a graph

Previous Work: speed up the RWR computation by preprocessing the graph but have limited scalability

Our Method: partition the adjacency matrix into small submatrices using hub-and-spoke structure and compute RWR accurately from the submatrices using block elimination

Our Result: outperform state-of-the-art methods in terms of speed, space efficiency, and accuracy

Our Contributions
Propose an exact method (BEAR-Exact) and an approximate method (BEAR-Approx) with the following properties compared with their respective state-of-the-art competitors:

- Fast: up to 8X less query time and 12X less preprocessing time (BEAR-Exact); a better time/accuracy trade-off (BEAR-Approx)
- Space efficient: up to 22X less memory requirements (BEAR-Exact); a better space/accuracy trade-off (BEAR-Approx)
- Accurate: guaranteed exactness (BEAR-Exact); a better trade-off between accuracy, time, and space (BEAR-Approx)
- Versatile: applicable to belief propagation, personalized PageRank, and effective importance

Definitions
Random Walk with Restart (RWR)
- Goal: measure relevance between two nodes in a graph
- Given: a graph \( G \), a seed node \( s \), and restart probability \( c \)
- Compute: stationary probability that a random surfer stays at each node
- Mathematical Definition: find \( r \) satisfying
  \[
  r = (1-c) \hat{A}r + cq
  \]
  \[
  \Leftrightarrow (1 - (1-c)\hat{A})r = cq
  \]
  \[
  \Leftrightarrow Hr = cq
  \]

where
- \( A \in \mathbb{R}^{n \times n} \) (input): row-normalized adjacency matrix
- \( q \in \mathbb{R}^n \) (input): query vector where \( q_i = 1 (i = s) \)
- \( r \in \mathbb{R}^n \) (output): RWR scores where \( r_i \) is relevance between \( i \) and \( s \)

Previous Methods
- Background: RWR score vector \( r \) is computed with regard to many different query vector \( q \)
- Approaches: preprocess the adjacency matrix to speed up the query processing (i.e., computing \( r \) w.r.t. given \( q \))
- Limitations: produce large dense precomputed matrices or do not guarantee exactness

1. Inversion
   Exact, \#nz=627M
2. QR (Fujiiwara et al. 12)
   Exact, \#nz=428M
3. LU (Fujiiwara et al. 12)
   Exact, \#nz=10M
4. B_LIN (Tong et al. 07)
   Approx, \#nz=8M
5. NB_LIN
   Approx, \#nz=3M
6. BEAR-Exact (Proposed)
   Exact, \#nz=0.4M

Preprocessing step (one-time cost)
1. Reordering: reorder \( H \) so that it has a large block-diagonal submatrix
2. Partitioning: partition \( H \) into submatrices
3. Schur Complement: compute the Schur complement \( S \) of \( H_{11} \)
4. Inverting: precompute the inversion of \( H_{11} \) and \( S \)

Fast and space efficient since the partitioned and precomputed matrices are small

Query step (repetitive cost)
Compute the RWR score vector \( r \) w.r.t. a given query vector \( q \) from the precomputed matrices using block elimination

Further optimization:
1. Replace \( H_{11}^{-1} \) and \( S^{-1} \) with their LU-decomposed matrices (sparser)
2. (BEAR-Approx) Drop near-zero entries from the precomputed matrices

Experiments
Binary codes and datasets are available at http://kdmlab.org/bear

- BEAR-Exact: superior scalability with up to 22X less memory requirements – Figure (a)
- up to 12X less preprocessing time – Figure (b)
- up to 8X less query time – Figure (c)

- BEAR-Approx: 250X faster and 50X less space with higher accuracy – Figures (d) and (e)

(a) Space requirements
(b) Preprocessing time
(c) Query time
(d) Accuracy vs. Speed
(e) Accuracy vs. Space requirements