Fast Flow-based Random Walk with Restart in a Multi-query Setting

Yujun Yan* Mark Heimann* Di Jin* Danai Koutra*

Abstract

As graph datasets grow, faster data mining methods become indispensable. Random Walk with Restart (RWR), belief propagation, semi-supervised learning, and more graph methods can be expressed as a set of linear equations. In this work, we focus on solving such equations fast and accurately when large number of queries need to be handled. We use RWR as a case study, since it is widely used not only to evaluate the importance of a node, but also as a basis for more complex tasks, e.g., representation learning and community detection. We introduce a new, intuitive two-step divide-and-conquer formulation and a corresponding parallelizable method, FlowR, for solving RWR with two goals: (i) fast and accurate computation under *multiple* queries; (ii) one-time message exchange between subproblems. We further speed up our proposed method by extending our formulation to carefully designed *overlapping* subproblems (FlowR-OV) and by leveraging the strengths of iterative methods (FlowR-Hyb). Extensive experiments on synthetic and real networks with up to ~ 8 million edges show that our methods are accurate and outperform in runtime various state-of-the-art approaches, running up to 34× faster in preprocessing and up to 32× faster in query time.

1 Introduction

As the amount of data that we generate daily grows rapidly, so does our need for scalable methods that can help analysts gain insights into it. Among the various tasks that apply to large graph data, Random Walk with Restart (RWR) is a frequently used, fundamental method that captures relative node importance and discovers local graph structures. It is a basic building block for numerous tasks, such as sampling in representation learning [8], ranking [20], community detection [1], used as a preprocessing step in other tasks, such as feature extraction [8] and link prediction.

RWR requires solving linear equations, as do other graph methods, such as semi-supervised learning, Belief Propagation [13, 6], SimRank [14], and more (cf. Table 1). Using RWR as a case study, we focus on the following problem, and propose methodology that can be generalized to other graph methods as well:

*University of Michigan, Ann Arbor

Table 1: Graph methods that can be expressed in the form $\mathbf{r} = \mathbf{E} \cdot \mathbf{r} + \mathbf{q}$. A and D denote the adjacency matrix and the diagonal degree matrix, resp.

Method	To-invert Matrix E	Input q	Notation
RWR [20]	$c\mathbf{A}\mathbf{D}^{-1}$	$(1-c)\mathbf{e}$	1-c: fly-out prob
Katz centr. [12]	$a'\mathbf{A}^T$	indicator vec. $\mathbf{e}$	a' weight for walks
(mincut) SSL [4]	$-a(\mathbf{D}-\mathbf{A})$	node labels $\mathbf{y}$	a: coupling strength
FaBP [13]	$\alpha\mathbf{D}-c'\mathbf{A}$	node beliefs $\boldsymbol{\phi}$	α, c': homophily
LinBP [6]	$\mathbf{H}^2 \otimes \mathbf{D} - \mathbf{H} \otimes \mathbf{A}$	$vec(\boldsymbol{\Phi})$	$\mathbf{H}$: influence matrix
SimRank [14]	$c"(\mathbf{A}\mathbf{D}^{-1}) \otimes (\mathbf{A}\mathbf{D}^{-1})$	$(1-c")vec(\mathbf{I})$	c'': decay factor

Problem 1. *In a multi-query setting, given k partitions of a graph G, we seek to* ***speed up*** *the solution of graph methods of the form:*

$$\mathbf{r} = \mathbf{E} \cdot \mathbf{r} + \mathbf{q}, \tag{1.1}$$

where $\mathbf{r}$ is the output vector, matrix $\mathbf{E}$ captures various structural properties of the graph, and $\mathbf{q}$ is a query.

At a large scale, exact matrix inversion (potentially coupled with LU or QR decomposition) cannot be applied directly due to its complexity. Often iterative methods are used instead. Most iterative methods [7, 16, 22, 21] compute multiple matrix-vector multiplications to achieve speedup. Other iterative methods belong to domain decomposition methods, such as Block Jacobi [22] and Additive Schwarz [16], and GraphLab [15]. The idea is to split the data into different clusters and compute the results *iteratively* using the latest values from neighbor clusters, thus requiring frequent cross-cluster communication until *all* clusters converge. Iterative methods, however, are inefficient for large numbers of queries since they require redundant re-computation per query, and many of them suffer from ill-conditioned pre-conditioners or weak convergence guarantees. Non-iterative methods [20, 10, 9] often give approximate solutions without guarantees, or rely on specialized decomposition methods.

To address these drawbacks, we propose a parallelizable, efficient, and accurate two-step approach, FlowR (Flow-based RWR), which can handle very large numbers of RWR queries with low response time during the online query step (second step) by performing an offline preprocessing step (first step). Intuitively, our method treats each of the k input partitions as a 'black box' with *input* signals from the other clusters and *output* signals from that cluster to the others. In this setting and unlike the formulation of decomposition methods,

each *input* signal is linked to a node rather than an edge. The solution has two components: (i) the solution when there are no cross-edges between partitions, which can be obtained independently with only local information, and (ii) the solution that 'responds' to the input signals that flow into nodes with cross-cluster connections. The input signals depend on the cluster connection topology (which can be precomputed and stored to avoid re-computation) and also on the query vectors.

Additionally, we propose a new strategy using overlapping clusters to reduce the computational cost. Unlike prior work [2] that aims to minimize the communication *frequency*, our approach targets the minimization of boundary nodes. Although this resembles the vertex-cover problem, it differs in that some boundary nodes may be counted multiple times.

Our contributions can be summarized as:

• **Theoretical Formulation**: We introduce a new two-step formulation for efficiently solving graph methods in multi-query settings, and extend it to overlapping partitions for further speedup in computation.

• **Fast Algorithms**: We propose FlowR, a fast two-step algorithm for the case study on RWR, and its overlapping counterpart, FlowR-OV. We also present a hybrid variant, FlowR-Hyb, which combines the benefits of iterative methods and our overlapping method for even better performance.

• **Experiments**: We perform extensive experiments on real-world networks with up to 7.6 M edges to test the runtime and accuracy of the proposed methods. We show that our methods outperform the state-of-the-art approaches by up to 34× in the preprocessing stage and 32× in the query stage.

The full paper (including the appendices with proofs, additional experiments, and discussion), the data, and the code for this work are publicly available at `https://github.com/Yujun-Yan/LINSYS`.

2 Related Work

Our work is related to the following topics: iterative graph methods, distributed computations, and clustering/partitioning (discussed in Appendix B).

Iterative Graph Methods. Many graph methods involve solving a linear system, such as computation of hitting and commute times, MaxCut, random spanning tree, and nearest tree, all the cases that we give in Table 1, and more. A category of iterative methods for linear equations is based on Krylov subspace methods. This category includes several methods that we use as baselines in our experiments [18]: Generalized Minimum Residual Method (GMRES), BiConjugate Gradient Stabilized Method (BiCGSTAB), Conjugate Gradient Squared Method (CGS) and Transpose-Free QMR Method (TFQMR). These generally suffer from slow query response time due to redundant re-computations for each query, and many often do not converge. For other iterative methods, we refer the reader to the introduction. Unlike these works, FlowR and its variants avoid redundant computations, combine the strengths of block elimination and Additive Schwarz methods, and outperform these approaches in most cases.

Distributed Graph Computations. Most distributed methods focus on PageRank and random walks due to their wide range of applications. In general, these methods exploit the block structure of many real-world graphs [20] and find approximate solutions. BEAR [10], speeds up random walks by leveraging block elimination and a specialized clustering method tailored to networks with skewed degree distributions. On the message-passing front, GraphLab [15] is a general, vertex-centric framework for a variety of algorithms on graphs, but requires continuous and costly synchronization between clusters and results in approximate solutions. To reduce the communication overhead, it also employs asynchronous communication. Andersen et al. [2] have shown that trading off some storage and allowing for overlapping clusters can reduce communication for iterative methods. Unlike these works, our solution has a simple interpretation (system with input/output signals), gives exact solutions, and involves a one-time communication between clusters in a distributed environment.

Table 2: Qualitative comparison of approaches: (1)-(2) potential for exact and approximate solutions; (3)-(4) compatibility with partitions and overlapping clusters; (5) interpretability; (6) convergence guarantees. '?' means dependence on the exact method.

	Exact	*Approx.*	*Partitions*	*Overlap*	*Interpr.*	*Converg.*
Krylov subspace [18]	✗	✓	✗	✗	✓	?
Decomp. (iter.) [22, 16, 15]	✗	✓	✓	✓	✗	?
BEAR [10]	✓	✓	✓	✗	✗	✓
FlowR	✓	✓	✓	✓	✓	✓

3 Notations & Definitions

Let $G(\mathcal{V}, \mathcal{E})$ be a graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, organized in k clusters C_i. Its edges are organized in an adjacency matrix $\mathbf{A}$, and its node degrees in a diagonal matrix $\mathbf{D}$. In the general form, we will refer to a transformed graph represented by matrix $\mathbf{E}$, which is a function of the adjacency and degree matrices (Table 1). For RWR, $\mathbf{E} \propto \mathbf{A}\mathbf{D}^{-1}$ is related to the transition matrix. Table 3 gives the most common symbols. Next, we introduce important concepts for our formulation.

Table 3: Major symbols and definitions. Capital bold letters for matrices; small bold font for vectors; small regular font for scalars.

Notation	Description
$G(\mathcal{V}, \mathcal{E})$	graph
$\mathcal{V}$, $n = \lvert\mathcal{V}\rvert$	node-set and number of nodes of G, resp.
$\mathcal{E}$, $m = \lvert\mathcal{E}\rvert$	edge-set and number of edges of G, resp.
C_i	cluster or partition i
$\partial\mathcal{V}$, $\partial\mathcal{V}_i$	set of boundary nodes for G and C_i, resp.
$\mathcal{V}^{in}$, $\mathcal{V}^{in}{}_i$	set of internal nodes for G and C_i, resp.
$\mathbf{A}$	adjacency matrix of G, with entries $a_{(ij)}$
$\mathbf{D}$	diagonal degree matrix, $d_{(ii)} = \sum_j a_{(ij)}$ & $d_{(ij)} = 0$ for $i \neq j$
$\mathbf{I}$	identity matrix
$\mathbf{E}, \mathbf{E}_{ij}$	structual related matrix and cross-cluster matrix (for C_i and C_j)
$\widetilde{\mathbf{E}}_{ij}$	non-zero submatrix of the cross-cluster matrix $\mathbf{E}_{ij}$
$\mathbf{Z_i}$	non-singular matrix corresponding to submatrix i, $= [\mathbf{I} - \mathbf{E}_{ii}]^{-1}$
$\mathbf{r}$	solution or posterior values or output
$\mathbf{q}$	query or prior values or input
k	# of clusters or communities or patterns
t	# of iterations
$[\mathbf{r}_1; \mathbf{r}_2]$	vertically stacking column vectors $\mathbf{r}_1, \mathbf{r}_2$
subscript i	reference to submatrix or subvector corresponding to cluster C_i
subscript (i)	reference to node i
$\lvert\cdot\rvert$	size (number of elements) of set or vector $\cdot$

DEFINITION 1. (BOUNDARY NODES ($\partial\mathcal{V}$)) *The boundary nodes of G are defined as*

$$\partial\mathcal{V} = \{u \in \mathcal{V} : \exists (u, v) \in \mathcal{E} \text{ s.t. } u \in C_i; v \notin C_i\}.$$

Put simply, the boundary nodes are adjacent to edges that cross partitions. We will refer by $\partial\mathcal{V}_i$ to the boundary nodes of C_i.

DEFINITION 2. (INTERNAL NODES ($\mathcal{V}^{in}$)) *The internal nodes of G are defined as $\mathcal{V}^{in} = \{u \in \mathcal{V} \setminus \partial\mathcal{V}\}$, where $\partial\mathcal{V}$ represents the boundary nodes. The internal nodes of cluster C_i are denoted by $\mathcal{V}_i^{in}$.*

EXAMPLE 1. *In Fig. 2, the graph on the left is organized in two clusters, C_A and C_B, and has 3 boundary nodes: 2, 3, and 7. All the other nodes are internal nodes.*

Vector and Matrix Representation: The output and input vectors $\mathbf{r}$ and $\mathbf{q}$ are stacked by their constituent vectors per cluster, i.e., $\mathbf{r} = [\mathbf{r}_1; \mathbf{r}_2; \ldots; \mathbf{r}_k]$ and $\mathbf{q} = [\mathbf{q}_1; \mathbf{q}_2; \ldots; \mathbf{q}_k]$. Furthermore, these constituent vectors $\mathbf{r}_i$ are organized such that the entries corresponding to the internal nodes of the cluster come before the boundary nodes, i.e., $\mathbf{r}_i = [\mathbf{r}_i^{in}; \mathbf{r}_i^{\partial\mathcal{V}}]$ and $\mathbf{q}_i = [\mathbf{q}_i^{in}; \mathbf{q}_i^{\partial\mathcal{V}}]$. We reorder the matrix of structural properties $\mathbf{E}$ in a similar way, as shown in Fig. 1. Next, we define its submatrices.

DEFINITION 3. (CROSS-CLUSTER MATRIX) *The cross-cluster matrix $\mathbf{E}_{ij} = diag(\mathbf{0}, \widetilde{\mathbf{E}}_{ij})$, where $\widetilde{\mathbf{E}}_{ij}$ is $\lvert\partial\mathcal{V}_i\rvert \times \lvert\partial\mathcal{V}_j\rvert$ $(i \neq j)$ in Fig. 1, stores the connections between clusters C_i and C_j $(i \neq j)$ and is a submatrix of $\mathbf{E}$.*

Each matrix is ordered so that its internal nodes $\mathcal{V}_i^{in}$ precede its boundary nodes $\partial\mathcal{V}_i$. Since cross-edges only exist between boundary nodes, the cross-cluster matrices have only one non-zero block on the diagonal. For completeness, we define $\widetilde{\mathbf{E}}_{ii} = \mathbf{0}$.

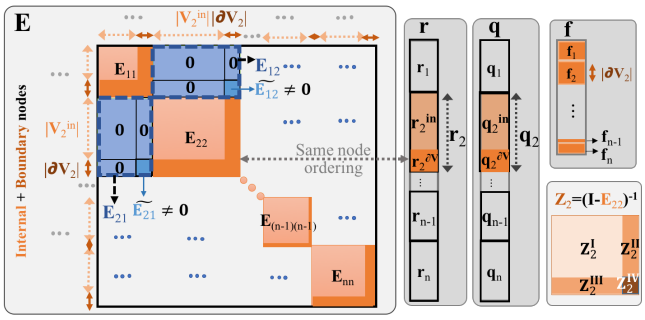


Figure 1: Vector and Matrix Representations. Entries ordered by cluster and node type (internal nodes first, then boundary nodes).

DEFINITION 4. (CLUSTER-MATRIX 'INVERSE') *We define the cluster-matrix 'inverse' as: $\mathbf{Z}_i = [\mathbf{I} - \mathbf{E}_{ii}]^{-1}$, where $\mathbf{E}_{ii}$ is the submatrix corresponding to cluster C_i. We organize it as shown in Fig. 1.*

An important notion in our derivation is the 'flow' (input) to a cluster C_i, which captures the one-time cross-cluster messages. We denote as $\mathbf{f}_{(i)}$ the 'flow of communications' (or flow) to each boundary node i, i.e., the scalar input values from its neighbors that belong to other clusters. Flow is dependent on the graph structure and the query. We note that our definition of flow differs from other definitions [17] in distributed settings that quantify communication *volume*; in our case it just captures the *values* of the computations performed from neighboring nodes in other clusters.

DEFINITION 5. (FLOW $\mathbf{f}_i$ OF CLUSTER C_i) *The flow (column) vector $\mathbf{f}_i$ of cluster C_i consists of its boundary nodes' flows, as shown in Fig. 1. Mathematically, the flow vector $\mathbf{f}_i$ of each cluster C_i is defined as: $\mathbf{f}_i = \sum_{i \neq j, j=1}^{k} \widetilde{\mathbf{E}}_{ij} \mathbf{r}_j^{\partial\mathcal{V}}$.*

The total flow for the graph is obtained by stacking the flow vectors of the k clusters: $\mathbf{f} = [\mathbf{f}_1; \mathbf{f}_2; \ldots; \mathbf{f}_k]$.

EXAMPLE 2. *In Fig. 2, the flows characterize the **total one-time impact** of node 7 to node 2, 3, and the **total impact** of nodes 2, 3 to node 7. This is **different from traditional message passing**, where messages iteratively pass between clusters.*

4 FlowR for Distinct Clusters

In the following sections, we present the main results of our proposed formulation, and introduce our algorithm, FLOWR. For detailed derivations, we refer the interested reader to the supplementary file (Appendix A).

4.1 Intuition and Key Result.

Our key idea is to find a two-step solution for equations of the form of Eq. (1.1), which include our target application RWR. Given k non-overlapping clusters of matrix $\mathbf{E}$ (or graph

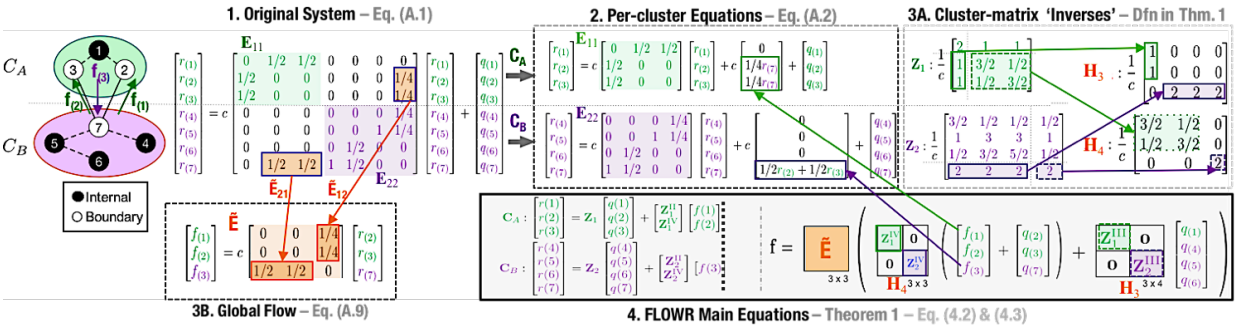


Figure 2: FlowR: ordered computations on a small graph with two clusters (C_A in green, C_B in purple). Subscripts in parentheses refer to nodes and the others to clusters. To solve RWR (after pre-computations), we (1) find the global flow in Eq. (4.3) and (2) use it to solve the per-cluster Eq. (4.2), in parallel.

G), we analytically decompose the solution into two parts. The first part represents the solution without accounting for flow across connections between clusters, and can be computed in parallel. Since most real-world graphs have many inter-cluster connections, we need a second part to 'correct' for the previously ignored cross-cluster communication (part 2). The 'correction' is associated with the flows into boundary nodes. In order to compute the 'correction' term in parallel, we need to first solve for flows via a reduced system that captures the relationship between them.

Theorem 1.[FLOWR] The original linear system $\mathbf{r} = \mathbf{E}\mathbf{r}+\mathbf{q}$ is equivalent to the following two equations:

$$\begin{bmatrix}\mathbf{r}_i^{in}\\ \mathbf{r}_i^{\partial\mathcal{V}}\end{bmatrix} = \mathbf{Z}_i \begin{bmatrix}\mathbf{q}_i^{in}\\ \mathbf{q}_i^{\partial\mathcal{V}}\end{bmatrix} + \begin{bmatrix}\mathbf{Z}_i^{\mathrm{II}}\\ \mathbf{Z}_i^{\mathrm{IV}}\end{bmatrix} \mathbf{f}_i, \quad \forall \text{ cluster } i = 1,\ldots,k \tag{4.2}$$

where the $|\partial\mathcal{V}|\times 1$ global flow $\mathbf{f}$ between the boundary nodes satisfies the condition:

$$\mathbf{f} = \widetilde{\mathbf{E}} \cdot \left(\mathbf{H_4}\cdot(\mathbf{f}+\mathbf{q}^{\partial\mathcal{V}}) + \mathbf{H_3}\cdot\mathbf{q}^{in}\right) \tag{4.3}$$

where $\widetilde{\mathbf{E}}$ is a block matrix induced on the boundary nodes (with $\widetilde{\mathbf{E}}_{ij}$ as sub-matrices), $\mathbf{H_4} = diag(\mathbf{Z}_1^{\mathrm{IV}},\ldots,\mathbf{Z}_k^{\mathrm{IV}})$ and $\mathbf{H_3} = diag(\mathbf{Z}_1^{\mathrm{III}},\ldots,\mathbf{Z}_k^{\mathrm{III}})$. Both $\widetilde{\mathbf{E}}$ and $\mathbf{f}$ are arranged by cluster.

Proof. See Appendix A of the supplementary material.

Intuitively, Eq. (4.2) shows the exact decomposition of the solution: the first term represents the case when no connections exist between clusters; the second term represents the correction for the cross-cluster flows. Equation (4.3) defines a reduced-size linear system that determines the flow between the boundary nodes of different clusters. Assuming that the number of boundary nodes is small compared to the total number of nodes in the original graph, this system can be solved efficiently. Once Eq. (4.3) is solved, it can be substituted in Eq. (4.2), which can then be solved independently (and in parallel) for each cluster. Given that each of the linear systems is significantly smaller than the original one and the majority of them can be solved in parallel in a distributed environment, this approach is expected to yield significant speedup, without trading off accuracy.

4.2 Proposed Algorithm: FlowR. Based on our analysis, we propose an efficient algorithm, Alg. 1, for solving linear equations in a multi-query setting. Following prior work [10], our method, FLOWR, can be split into (i) an offline preprocessing stage that does not depend on the input vector and focuses on matrix reordering and the storage of intermediate matrices, and (ii) an online and fast query stage that solves the linear system for a given query $\mathbf{q}$ by finalizing the computation of the main FLOWR equations (i.e., Eq. (4.2) and (4.3)).

In FLOWR, we assume that the k input clusters are non-overlapping, while in Sec. 5 we take advantage of the benefits of overlapping clusters in distributed computations, and propose a strategy for minimizing the boundary nodes. In Line 6 and 10, `LU_invert` involves first LU decomposition and then the inversion of the L and U matrices. This step can be replaced by other approximate methods for further speedup.

Algorithm 1 FLOWR

1: **// 1: Preprocessing Stage**
2: **Input**: graph $G(\mathcal{V},\mathcal{E})$, structure-based matrix $\mathbf{E}'$, k clusters: $\{C_1,\ldots,C_k\}$
3: **Output**: $\widetilde{\mathbf{E}}$, $\mathbf{H_3}$, $\mathbf{T}$, $\mathbf{T_1}$
4: $\mathbf{E}$ = reorder($\mathbf{E}'$) ▷ as shown in Fig. 1
5: For $i = 1,\ldots,k$
6: $\quad\mathbf{Z_i}$ = LU_invert($[\mathbf{I}-\mathbf{E}_{ii}]$) ▷ parallelizable
7: $\mathbf{H_3} = diag(\mathbf{Z_1}^{\mathrm{III}},\ldots,\mathbf{Z_k}^{\mathrm{III}})$ ▷ matrices shown in Fig. 1
8: $\mathbf{H_4} = diag(\mathbf{Z_1}^{\mathrm{IV}},\ldots,\mathbf{Z_k}^{\mathrm{IV}})$
9: $\mathbf{T_1} = \widetilde{\mathbf{E}}\cdot\mathbf{H_4}$ ▷ $\widetilde{\mathbf{E}}$= mat $\mathbf{E}$ induced on boundary nodes
10: $\mathbf{T}$ =LU_invert($\mathbf{I}-\mathbf{T_1}$) ▷ from Eq. (4.3)
11: **// 2: Query Stage**
12: **Input**: query $\mathbf{q}$, graph $G(\mathcal{V},\mathcal{E})$, $\widetilde{\mathbf{E}}$, $\mathbf{H_3}$, $\mathbf{T}$, $\mathbf{T_1}$
13: **Output**: $\mathbf{r}$
14: $\mathbf{y_r}$ =reorder($\mathbf{q}$) ▷ as shown in Fig. 1
15: $\mathbf{b} = \mathbf{T_1}\mathbf{q}_r^{\partial\mathcal{V}} + \widetilde{\mathbf{E}}(\mathbf{H_3}\mathbf{q}_r^{in})$
16: $\mathbf{f} = \mathbf{T}\cdot\mathbf{b}$ ▷ compute flow from Eq. (4.3)
17: $\mathbf{r}_r$=Eq. (4.2) ▷ compute the result per cluster
18: $\mathbf{r}$ =map($\mathbf{r}_r$) ▷ map back to original node IDs and get the final result

5 FlowR-OV for Overlapping Clusters

As shown in Appendix D of the supplementary material, the dominant factor in the runtime of FlowR is the number of flows during the preprocessing step (or the size of $\widetilde{\mathbf{E}}$, which is equal to the number of boundary nodes in the non-overlapping case). To further speed up the solution of Eq. (1.1), we propose a method, FlowR-OV, that aims to minimize the number of flows by replicating selected nodes in the appropriate clusters (thus resulting in overlapping clusters). Our optimization differs from edge cut minimization (the typical goal in clustering), and the vertex-cut problem (some boundary nodes are counted multiple times due to replication). Inspired by [2], which uses replication to reduce cross-cluster communication frequency, we propose a new replication strategy to achieve a different goal, namely flow minimization.

Next, we (i) define concepts specific to the overlapping case, (ii) extend our formulation to this case and introduce FlowR-OV, (iii) propose a fast node replication technique for flow minimization, and (iv) speed up the solution via a hybrid approach that combines the strengths of FlowR-OV and iterative approaches.

5.1 Definitions. In the overlapping case, boundary nodes are further divided into *replicated* and *non-replicated* nodes, while internal nodes are split into *overhead* and *pure internal* ones. This distinction matters because overhead nodes do not contribute to the number of flows (as boundary nodes do), but they are still involved in more computations than pure internal nodes. Fewer flows can reduce the preprocessing and query time if the overhead nodes are controlled. Intuitively, if a node is replicated from cluster C_j to C_i, its neighbors in both clusters will become *overhead* nodes (converted from boundary in C_i or internal nodes in C_j). Formally:

Definition 6. (Replicated boundary nodes ($\partial\mathcal{V}^r$)) *The replicated boundary nodes of G are defined as $\partial\mathcal{V}^r = \{u \in \partial\mathcal{V}$ and $\exists i \in \{1,\dots,k\}$ with $i \neq j : u \in C_i; u \in C_j\}$.*

Definition 7. (Overhead nodes (ov)) *The overhead nodes of G are the internal nodes that are connected to at least one replicated boundary node (or an instance thereof) in $\partial\mathcal{V}^r$: $\mathcal{V}^{ov} = \{u \in \mathcal{V}^{in} : \exists v \in \partial\mathcal{V}^r$ s.t. $(u,v) \in \mathcal{E}; u,v \in C_i$ for some $i\}$.*

Definition 8. (Pure Internal nodes (pin)) *The pure internal nodes of G are its internal nodes that are not overhead nodes, i.e., $\mathcal{V}^{pin} = \{u \in \mathcal{V}^{in} \setminus \mathcal{V}^{ov}\}$.*

Example 3. *In Fig. 3, if we replicate node 7 to Cluster A (i.e., $\partial\mathcal{V}^r = \{7\}$, then nodes 2 and 3 become overhead nodes (from boundary), and nodes 4 and 5 in Cluster B also become overhead nodes (from internal). Also, in this case $\partial\mathcal{V} = \{7\}$.*

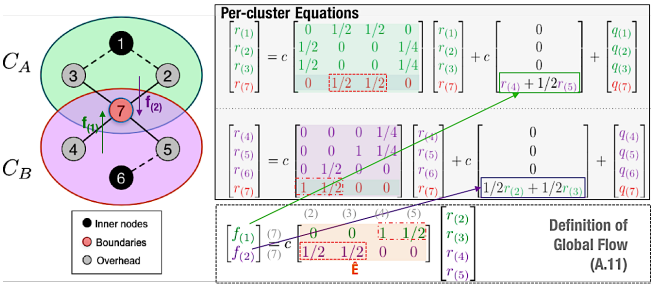


Figure 3: Example: FlowR-OV on two clusters.

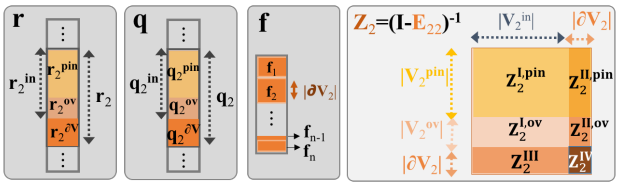


Figure 4: Vector and matrix representations for overlapping clusters.

We use similar vector and matrix representation that we presented in Sec. 4, but the internal nodes are further organized in subtypes as shown in Fig. 4.

The previous definition of flow is extended to $\mathbf{f}_i = \sum_{i\neq j, j=1}^{k} \widetilde{\mathbf{E}}_{ij}\mathbf{r}_j^{\partial\mathcal{V}+ov}$, as the boundary nodes also receive input signals from overhead nodes in other clusters.

5.2 Proposed Formulation. The general idea is similar to the non-overlapping case. We treat each cluster C_i as a small system and compute the flows before solving each small system independently. Based on a carefully designed replication policy (discussed below), we reduce the number of boundary nodes by turning a subset of them into overhead nodes. Then, by extending Theorem 1 to include the new node definitions, we obtain the main equations of FlowR-OV. Figure. 5 shows the preprocessing bottlenecks for FlowR and FlowR-OV, and the reduction of the linear system size for the latter.

Theorem 2.[FlowR-OV] In the overlapping case, the linear system $\mathbf{r} = \mathbf{E}\mathbf{r} + \mathbf{q}$ is equivalent to the following two equations:

(5.4)
$$\begin{bmatrix}\mathbf{r}_i^{pin}\\ \mathbf{r}_i^{ov}\\ \mathbf{r}_i^{\partial\mathcal{V}}\end{bmatrix} = \begin{bmatrix}\mathbf{Z}_i^{\text{II,pin}}\\ \mathbf{Z}_i^{\text{II,ov}}\\ \mathbf{Z}_i^{\text{IV}}\end{bmatrix}\mathbf{f}_i + \mathbf{Z}_i\begin{bmatrix}\mathbf{q}_i^{pin}\\ \mathbf{q}_i^{ov}\\ \mathbf{q}_i^{\partial\mathcal{V}}\end{bmatrix}, \quad \forall \text{ cluster } i = 1,\dots,k$$

where the global flow $\mathbf{f}$ between the boundary nodes satisfies the condition:

(5.5)
$$\mathbf{f} = \hat{\mathbf{E}}\cdot\Big(diag\Big(\begin{bmatrix}\mathbf{Z}_1^{\text{II,ov}}\\ \mathbf{Z}_1^{\text{IV}}\end{bmatrix},\dots,\begin{bmatrix}\mathbf{Z}_k^{\text{II,ov}}\\ \mathbf{Z}_k^{\text{IV}}\end{bmatrix}\Big)(\mathbf{f} + [\mathbf{q}_1^{\partial\mathcal{V}};\dots;\mathbf{q}_k^{\partial\mathcal{V}}]) + diag\Big(\begin{bmatrix}\mathbf{Z}_1^{\text{I,ov}}\\ \mathbf{Z}_1^{\text{III}}\end{bmatrix},\dots,\begin{bmatrix}\mathbf{Z}_k^{\text{I,ov}}\\ \mathbf{Z}_k^{\text{III}}\end{bmatrix}\Big)\mathbf{q}^{in}\Big).$$

$\hat{\mathbf{E}}$ is the induced matrix $\mathbf{E}$ on $\partial\mathcal{V}$ and $\mathcal{V}^{ov}$.

Proof. See Appendix A of the supplementary material.

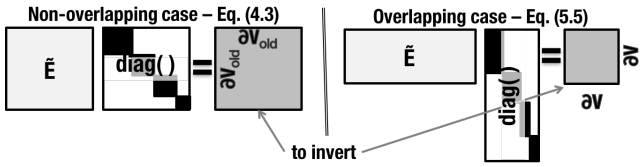


Figure 5: Preprocessing barrier of FlowR vs. FlowR-OV. FlowR-OV leads to a smaller linear system.

Although replicated nodes have multiple flows (one per cluster in the overlap), perhaps counter-intuitively **the flow size is smaller than in the non-overlapping case**. For example, in Fig. 3, the number of flows is reduced from 3 (in the non-overlapping case) to 2 in the overlapping case (one flow for node 7 in each of its clusters). At a high level, we only replicate boundary nodes that reduce the size of flow by converting other boundary nodes to overhead nodes (cf. policy below). Let $\partial\mathcal{V}^{old}$ be the boundary nodes in the non-overlapping case. By denoting the set of overhead nodes that were previously boundary nodes as $\{\partial\mathcal{V}^{old} \rightarrow \mathcal{V}^{ov}\}$, we can express the set of boundary nodes in the overlapping case as $\partial\mathcal{V} = \partial\mathcal{V}^{old} \setminus \{\partial\mathcal{V}^{old} \rightarrow \mathcal{V}^{ov}\}$. We omit the algorithm since it is similar to Alg. 1. The main changes include (i) the new vector and matrix representations; (ii) $\hat{\mathbf{E}}$ instead of $\widetilde{\mathbf{E}}$ in Alg. 1 (line 9); and (iii) extended definitions for $\mathbf{H_3}$ and $\mathbf{H_4}$ (lines 7 & 8).

5.3 Proposed Policy. We conjecture that finding a clustering method which minimizes the boundary nodes while imposing a maximum threshold for the size of each cluster is a hard problem, and provide details in Appendix C. Thus, we focus on providing a heuristic policy for replicating boundary nodes to create overlapping clusters. Intuitively, by focusing on one cluster at a time, our policy copies into that cluster the set of nodes from *other* clusters (i) which are connected to multiple nodes in C_i, and (ii) without which some boundary nodes in C_i would have been internal.

To illustrate this, we present the replication of boundary nodes from other clusters in cluster C_i. Let $deg^{cc}(v)$ be the number of cross-cluster edges of node v (i.e., its degree induced on the adjacency matrix of all the boundary nodes). For node u in C_j, we define $N^i_{1d}(u)$ as the number of its neighbors in cluster C_i ($j \neq i$) that have cross-cluster degree 1. Put differently, this is the set of u's neighbors (in cluster C_i) that would have been internal if it were not for their cross-cluster connection to u. Formally:

$$N^i_{1d}(u) = |\{v \in C_i : (u,v) \in \mathcal{E}, deg^{cc}(v) = 1\}|, \forall u \in C_j, j \neq i \tag{5.6}$$

Our policy copies all the boundary nodes $u \in C_j$ with $N^i_{1d}(u) > 1$ into cluster C_i (i.e., all the nodes whose replication eliminates at least two boundary nodes in C_i). For k clusters, this policy takes time $O(k|\partial\mathcal{V}| \max_i |C_i|)$ in the worst case, but $O(k|\partial\mathcal{V}|)$ in practice when most vertices do not have high degrees.

Algorithm 2 FlowR-Hyb- Query Stage (only)

1: **Input**: query $\mathbf{q}$, graph $G(\mathcal{V},\mathcal{E})$, $\hat{\mathbf{E}}$, $\mathbf{H_3}$, $\mathbf{T_1}$
2: **Output**: $\mathbf{r}$

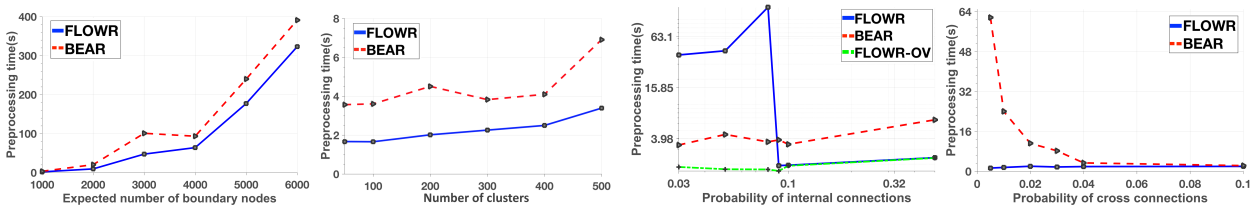
3: $\mathbf{y_r}$ =reorder($\mathbf{q}$) ▷ reorder y
4: $\mathbf{b} = \mathbf{T_1}\mathbf{q}_r^{\partial\mathcal{V}} + \hat{\mathbf{E}}(\mathbf{H_3}\mathbf{q}_r^{in})$ ▷
$\mathbf{H_3} = diag(\begin{bmatrix}\mathbf{Z}_1^{\mathrm{I,ov}}\\ \mathbf{Z}_1^{\mathrm{III}}\end{bmatrix}, \ldots, \begin{bmatrix}\mathbf{Z}_k^{\mathrm{I,ov}}\\ \mathbf{Z}_k^{\mathrm{III}}\end{bmatrix})$
5: $\mathbf{f}$ =PowerMethod($\mathbf{T_1}$,$\mathbf{b}$) ▷ Eq. (5.5)
6: $\mathbf{r}_r$=Eq. (5.4) ▷ compute the result per cluster
7: $\mathbf{r}$ =map($\mathbf{r}_r$) ▷ map to original node IDs for final results

5.4 Further Speedup via Hybrid Approach FlowR-Hyb. The bottleneck of our method in preprocessing is the LU decomposition and inversion for $\mathbf{T_1} = \hat{\mathbf{E}} \cdot (diag(\begin{bmatrix}\mathbf{Z}_1^{\mathrm{II,ov}}\\ \mathbf{Z}_1^{\mathrm{IV}}\end{bmatrix}, \ldots, \begin{bmatrix}\mathbf{Z}_k^{\mathrm{II,ov}}\\ \mathbf{Z}_k^{\mathrm{IV}}\end{bmatrix})$. However, sometimes, the size of $\mathbf{T_1}$ may not be sufficiently small, rendering the preprocessing of the decomposition and inverse costly (Alg. 1, line 10). Therefore, we introduce FlowR-Hyb, a hybrid method that only *precomputes* $\mathbf{T_1}$ and defers Eq. (5.5) to be solved *iteratively* in the query stage. This approach combines the benefits of iterative methods and FlowR-OV (lower preprocessing *and* query time, respectively). Algorithm 2 describes the modified query stage.

6 Experimental Analysis

We conduct experiments on both synthetic and real-world datasets to answer the following questions: **(i)** Is FlowR efficient? **(ii)** What is the benefit of FlowR-OV over FlowR? **(iii)** Is our proposed strategy effective? **(iv)** What is the benefit of FlowR-Hyb over FlowR-OV? **(v)** How do the methods perform with respect to baseline approaches?

Baselines. Direct is a simple baseline for solving Eq. (1.1) by directly computing the inverse of $[\mathbf{I}-\mathbf{E}]$, but it is not feasible in any of our datasets, except for `Net`. Instead, we compare our methods with BEAR, which is based on block elimination, and significantly outperforms QR decomposition and LU decomposition [10]. Although our methods are exact, we also consider 6 approximate, iterative approaches for RWR (Sec. 2): POWER (power method), GMRES, BiCGSTAB, CGS, TFQMR and MLDIVIDE, a method combined with Cholesky factorization and Gaussian Elimination provided by Matlab. Some of these methods often fail to converge. Although our method can be *easily parallelized*, considering our competitors are sequential, we use the *sequential version* to compare with them.



(a) Effect of boundary nodes **(b) Scalability** **(c) Min-cut partitions** **(d) Hub-spoke partitions**

Figure 6: Synthetic Experiments: Effect of different parameters on FlowR and BEAR.

Table 4: Real graph data from [19].

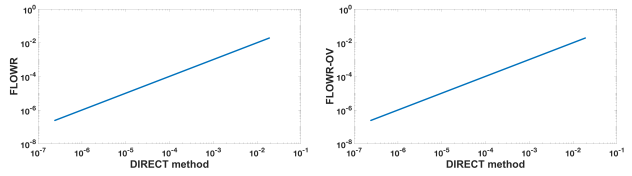
Name	Nodes	Edges	Description
Net	34 761	171 403	undir. Internet connection graph
Amazon	334 863	925 872	undirected co-purchase graph
RoadPA	1 088 092	1 541 898	undirected road network
RoadTX	1 379 917	1 921 660	undirected road network
Web-Stan	281 903	2 312 497	directed web graph
Google	875 713	5 105 039	directed web graph
Web-Berk	685 230	7 600 595	directed web graph

Experimental setup. We ran our experiments on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz, and 256GB memory. For RWR, we set the fly-out probability $1-c = 0.15$. In both stages, FlowR benefits from a small number of boundary nodes. Given that there are no such methods in the literature, among fast partitioning methods, we chose Louvain [3], which usually generates 10% fewer boundary nodes than METIS [11]. We choose the highest level from the hierarchical partition of Louvain. Note that although Louvain is not guaranteed to produce balanced clusters, it facilitates FlowR as long as a) no giant clusters are generated and b) the structural matrices of output clusters are invertible.For the iterative baselines (except GMRES) and our hybrid method FlowR-Hyb, we set the stopping condition as error $\leq 10^{-9}$ (convergence). GMRES's threshold is set to 10^{-6} due to convergence issues. To reduce variance, we average query time over 1000 queries and report this.

Data. The real datasets are described in Table 4. The synthetic graphs are characterized by the following parameters: the number of clusters k, the probability of a node being a boundary node P_b, the probability of two nodes having an in-cluster edge P_{in}, and the probability of two nodes having a cross-cluster edge P_{cross}. The number of nodes per cluster is generated by a uniform random variable ranging from 20 to 180. By default, the synthetic graphs mimic the community structure of real graphs with $P_{in} = 0.3$, $P_{cross} = 0.04$ and $P_b = 0.1$ (i.e., many intra-cluster and few inter-cluster connections, and small portion of boundary nodes).

6.1 Experiments on Synthetic Networks. In Fig. 6, we show how the computation time is affected by different factors. To study how boundary nodes affect FlowR and BEAR, we change the expected number of boundary nodes from 1000 to 6000 while keeping the size of the graph unchanged. To study the scalability of the methods, we change the number of clusters (thus more nodes) while leaving the number of boundary nodes unchanged (on average). To show that the min-cut partitions and the minimum-boundary partitions achieve their optima asynchronously, we vary the probability of two internal nodes forming an edge. To study how the hub-and-spoke like graphs affect the efficiency of our proposed and other methods, we control the expected in-cluster edges for internal nodes and expected cross-cluster edges for boundary nodes.

Due to the limitation in space, Figure 6 shows the preprocessing time because similar patterns are revealed for the query time. In Figure 6a, we observe that if the inherent number of boundary nodes increases, the computation time of both FlowR and BEAR will increase accordingly, with FlowR running consistently faster than BEAR (due to a smaller system of linear equations for FlowR). Figure 6b shows that the preprocessing time of both FlowR and BEAR changes slowly with the increase in the number of clusters, with FlowR having a smaller increase rate, or equivalently better scalability. Figure 6c shows that if the traditional clustering structure (more within-cluster edges than cross-cluster edges) is inverted (e.g. the probability of forming an internal edge is below 0.1 in our experiments), the partition method based on min-cut criterion is a bad choice for our algorithms. This is expected as fewer internal edges will mislead min-cut methods to split the clusters and create many unexpected boundary nodes. Inversely, this experiment also suggests that a method tailored to boundary node minimization is expected to produce better results for our methods. Finally, Figure 6d depicts how the graph structure can lead to poor performance for our competitors. We observe that when the probability of forming cross-edges is low, FlowR outperforms BEAR significantly. This is because the tailored clustering method (SlashBurn) in BEAR has difficulty detecting hub nodes when the boundary nodes have low degrees. FlowR overcomes this issue and maintains promising performance regardless of the the probability of forming cross-edges.



(a) FlowR vs. Direct **(b) FlowR-OV vs. Direct**
Figure 7: Our methods are exact and give identical scores to DIR.

6.2 Experiments on Real Networks

Accuracy of FlowR. To show that FlowR and FlowR-OV are accurate, we use the Direct method to compute the accurate RWR scores per node and compare with the scores of our (exact) proposed methods. Figure 7 shows FlowR and FlowR-OV's results on `Net` compared with the actual results. The scores of the FlowR variants are identical to those of the Direct method (the pairs of <FlowR scores, actual scores> lie on the $y = x$ line). The results are similar for other small datasets, but we omit the plots for brevity.

Runtime. In this experiment we evaluate the efficiency of our proposed method by comparing the variants of FlowR with the baselines in terms of both preprocessing time (excluding the clustering time) and query time. We run the variants of FlowR and all baselines on 6 real-world graphs in Table 4.

The preprocessing time is illustrated on the left in Fig. 8. Our first observation is that our proposed methods work best on most real-world datasets (`Amazon`, `RoadPA`, `RoadTX`, and `Google`). Specifically, FlowR-Hyb is 34× faster than BEAR on `Amazon` and 15× faster on `Google`. In addition, FlowR-OV is 6× faster than BEAR. Also, unlike FlowR-OV and FlowR-Hyb, BEAR runs out of memory for two datasets.

As shown in Table 5, the preprocessing time of our variants and BEAR is decided mainly by the number of critical nodes (boundary nodes for FlowR, flows for FlowR-OV, hub nodes for BEAR) and maximum cluster size (especially for FlowR-Hyb). In most real graphs, the maximum cluster size is smaller than that of other critical nodes, and thus FlowR-Hyb achieves best performance. Due to our efficient replication policy, FlowR-OV also achieves overall better results than BEAR, with the exception of `Web-Stan` and `Web-Berk`. These two graphs have hub-and-spoke structure, for which BEAR optimizes and thus produces fewer critical nodes than our methods. However, our methods are suitable for different kinds of graphs and produce stable and satisfying results. We believe that if the original partition aimed at minimizing boundary nodes, the performance of FlowR would be even better.

The right chart in Figure 8 illustrates the query time for different datasets and methods. We observe that FlowR outperforms all baselines with the smallest

Table 5: Summary of crucial nodes for BEAR, FlowR and FlowR-OV and FlowR-Hyb

	Number of critical nodes			
Datasets	**BEAR (hubs)**	**FlowR ($\|\partial\mathcal{V}\|$)**	**FlowR-OV ($\|\mathbf{f}\|$)**	**FlowR-Hyb ($\max\|C_i\|$)**
`Amazon`	53,440	64,915	57,360	15,517
`RoadPA`	154,496	19,241	18,119	7,740
`RoadTX`	153,069	18,417	17,522	8,525
`Web-Stan`	16,017	27,902	12,353	25,253
`Google`	117,250	40,726	35,668	14,622
`Web-Berk`	50,005	98,792	26,810	47,481

query time in four datasets and second best (very close to the best) in the remaining. Specifically, FlowR achieves up to 32× speedup in query time, when the baselines do *not* run out of memory and have *at least 60%* queries that converge. In many cases, baselines *fail* to complete or converge in the majority of queries. Through a carefully-designed precomputation stage, our methods are faster in the query stage and, thus, more efficient in multi-query settings.

We note that FlowR-OV achieves improved results due to replication: it runs faster than FlowR in both the preprocessing and query stages in four out of six datasets. For `Amazon`, by successfully reducing the boundary nodes, FlowR-OV can return a result while the non-overlapping method, FlowR, runs out of memory due to the large number of boundary nodes. This demonstrates that our replicating policy helps not only with speedup, but also with runtime memory cost.

As Fig. 8 shows, our hybrid method FlowR-Hyb effectively enhances both preprocessing and query performance of FlowR-OV, especially for graphs that take a long time in preprocessing. For instance, the preprocessing time is improved by $\sim 30\times$ for `Amazon`, and its speedup varies in the other datasets. Thus, FlowR-Hyb has the strength of iterative methods. At the same time, it has the strength of FlowR-OV: its query response is significantly faster than that of all iterative methods in five out of six datasets and beats BEAR in four datasets. We perform further analysis to support and explain the last claim in Appendix E.

7 Conclusions

In this paper, using RWR as a case study, we focus on fast, distributed solutions for a general form of linear systems that is widespread in graph methods, such as diffusion processes, semi-supervised learning, and more. We derive a two-step solution and propose FlowR to solve RWR *exactly* and efficiently. Motivated by the the benefits of data replication across clusters and those of iterative methods, we further extend our solution to these settings, and introduce two more efficient variants, FlowR-OV and FlowR-Hyb. Our experiments on large, real networks show that FlowR and FlowR-

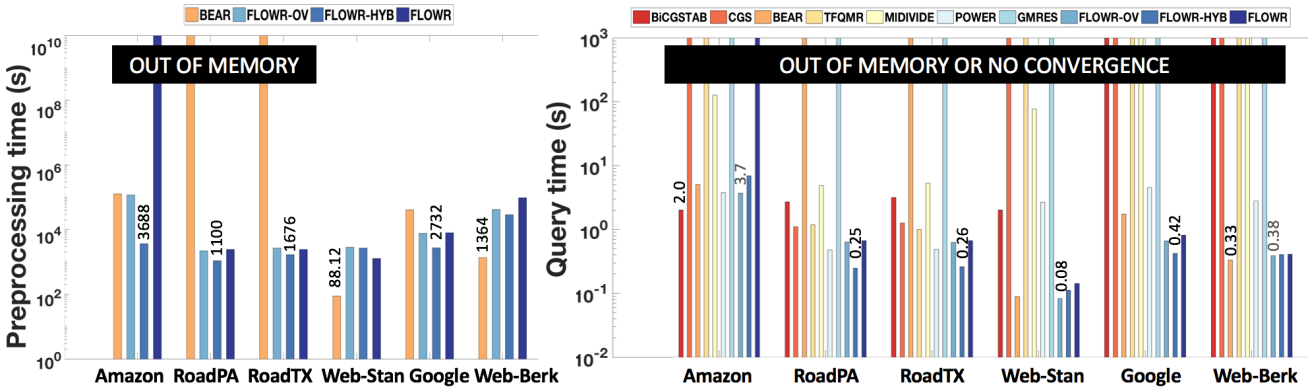


Figure 8: Comparisons of performance for different methods: FlowR-OV performs better or equally well in most real datasets.

OV outperform by up to 34× and 32× the best existing RWR approaches in preprocessing and query time, respectively (when these baselines do not run out of memory and have $> 60\%$ queries converge).

Our positive results show that our proposed method is promising even *without* operating on partitions with a minimum number of boundary nodes (which would be optimal). This opens up a new future direction: partitioning a graph to minimize the number of boundary nodes instead of edge cuts.

Acknowledgements: This material is based upon work supported by the University of Michigan.

References

[1] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.

[2] Reid Andersen, David F. Gleich, and Vahab Mirrokni. Overlapping Clusters for Distributed Computation. WSDM, pages 273–282. ACM, 2012.

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *JSTAT*, 2008(10):P10008, 2008.

[4] Avrim Blum and Shuchi Chawla. Learning from Labeled and Unlabeled Data Using Graph Mincuts. In *ICML*, pages 19–26. Morgan Kaufmann, 2001.

[5] Ulrik Brandes Daniel Fleischer. Vertex bisection is hard, too. *JGAA*, 13(2):119–131, 2009.

[6] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and Single-Pass Belief Propagation. *PVLDB*, 8(5):581–592, 2015.

[7] David F Gleich, Andrew P Gray, Chen Greif, and Tracy Lau. An inner-outer iteration for computing pagerank. *SISC*, 32(1):349–371, 2010.

[8] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864. ACM, 2016.

[9] Jinhong Jung, Namyong Park, Lee Sael, and U Kang. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *SIGMOD*, pages 789–804, 2017.

[10] Jinhong Jung, Kijung Shin, Lee Sael, and U Kang. Random Walk with Restart on Large Graphs Using Block Elimination. *ACM Trans. Database Syst.*, 41(2):12:1–12:43, 2016.

[11] George Karypis and Vipin Kumar. Multilevel k-way Hypergraph Partitioning. pages 343–348, 1999.

[12] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[13] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *ECML PKDD*, pages 245–260, 2011.

[14] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. Fast Computation of SimRank for Static and Dynamic Information Networks. In *EDBT*, pages 465–476. ACM, 2010.

[15] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *PVLDB*, 5(8):716–727, 2012.

[16] Ivo Marek and Daniel B Szyld. Algebraic schwarz methods for the numerical solution of markov chains. *Linear Algebra and its Applications*, 386:67–81, 2004.

[17] Sivasankaran Rajamanickam and Erik G Boman. Parallel partitioning with zoltan: Is hypergraph partitioning worth it? *Graph Partitioning and Graph Clustering*, 588:37–52, 2012.

[18] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[19] SNAP. http://snap.stanford.edu/data/index.html#web.

[20] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast Random Walk with Restart and Its Applications. In *ICDM*, pages 613–622, 2006.

[21] Abderezak Touzene. A new parallel block aggregated algorithm for solving markov chains. *J Supercomput*, 62(1):573–587, 2012.

[22] Yangbo Zhu, Shaozhi Ye, and Xing Li. Distributed pagerank computation based on iterative aggregation-disaggregation methods. In *CIKM*, pages 578–585, 2005.