Locality-Sensitive and Re-use Promoting Personalized PageRank Computations

Knowledge and Information Systems

Locality-Sensitive and Re-use Promoting Personalized PageRank Computations

--Manuscript Draft--

Manuscript Number:	KAIS-13-4561R2
Full Title:	Locality-Sensitive and Re-use Promoting Personalized PageRank Computations
Article Type:	Regular Paper
Keywords:	Personalized PageRank; Locality-Sensitivity; Reuse-Promotion; Scalability
Corresponding Author:	Jung Hyun Kim Arizona State University Tempe, AZ UNITED STATES
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Arizona State University
Corresponding Author's Secondary Institution:	
First Author:	Jung Hyun Kim
First Author Secondary Information:	
Order of Authors:	Jung Hyun Kim
	K. Selçuk Candan, Ph.D
	Maria Luisa Sapino, Ph.D
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	Node distance/proximity measures are used for quantifying how nearby or otherwise related two or more nodes on a graph are. In particular, personalized PageRank (PPR) based measures of node proximity have been shown to be highly effective in many prediction and recommendation applications. Despite its effectiveness, however, the use of personalized PageRank for large graphs is difficult due to its high computation cost. In this paper, we propose a Locality-sensitive, Re-use promoting, approximate Personalized PageRank (LR-PPR) algorithm for efficiently computing the PPR values relying on the localities of the given seed nodes on the graph: (a) The LR-PPR algorithm is locality sensitive in the sense that it reduces the computational cost of the PPR computation process by focusing on the local neighborhoods of the seed nodes. (b) LR-PPR is re-use promoting in that instead of performing a monolithic computation for the given seed node set using the entire graph, LR-PPR divides the work into localities of the seeds and caches the intermediary results obtained during the computation. These cached results are then reused for future queries sharing seed nodes. Experiment results for different data sets and under different scenarios show that LR-PPR algorithm is highly-efficient and accurate.

Under consideration for publication in Knowledge and Information Systems

Locality-Sensitive and Re-use Promoting Personalized PageRank Computations

Jung Hyun Kim[1], K. Selçuk Candan[1], and Maria Luisa Sapino[2]

[1]School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe AZ, USA; [2]Dipartimento di Informatica, University of Torino, I-10149 Torino, Italy

Abstract. *Node distance/proximity* measures are used for quantifying how *nearby* or otherwise *related* two or more nodes on a graph are. In particular, personalized PageRank (PPR) based measures of node proximity have been shown to be highly effective in many prediction and recommendation applications. Despite its effectiveness, however, the use of personalized PageRank for large graphs is difficult due to its high computation cost. In this paper, we propose a *Locality-sensitive*, *Re-use promoting*, *approximate Personalized PageRank* (LR-PPR) algorithm for efficiently computing the PPR values relying on the *localities* of the given seed nodes on the graph: (a) The LR-PPR algorithm is *locality sensitive* in the sense that it reduces the computational cost of the PPR computation process by focusing on the local neighborhoods of the seed nodes. (b) LR-PPR is *re-use promoting* in that instead of performing a *monolithic* computation for the given seed node set using the entire graph, LR-PPR divides the work into localities of the seeds and *caches* the intermediary results obtained during the computation. These cached results are then *reused* for future queries sharing seed nodes. Experiment results for different data sets and under different scenarios show that LR-PPR algorithm is highly-efficient and accurate.

Keywords: Personalized PageRank; Locality-Sensitivity; Reuse-Promotion; Scalability

1. Introduction

In many graph applications, how a given pair of nodes on a graph is related to each other is determined by the underlying graph topology. *Node distance/proximity* measures are commonly used for quantifying how *nearby* or otherwise *related to* two or more nodes on a graph are. Due to the wide-spread use of graphs in analysis, mining, and visualization of interconnected data, there are many definitions of the node distance and proximity. Path-length based definitions, such as those used by Palmer et al (2006), Boldi et al (2011), Cohen et al (2003), Wei (2010), Xiao et al (2009), Zhou et al (2009), are

Received xxx
Revised xxx
Accepted xxx

useful when the *relatedness* can be captured solely based on the properties of the nodes and edges on the *shortest* path (based on some definition of path-length). Random-walk based definitions, such as hitting distance (Chen et al, 2008; Mei et al, 2008) and personalized PageRank (PPR) score (Balmin et al, 2004; Chakrabarti, 2007; Jeh and Widom, 2002; Tong et al, 2006a; Tong et al, 2007; Liu et al, 2013; Lofgren et al, 2014; Maehara et al, 2014), of node relatedness, on the other hand, also take into account the density of the edges: intuitively, as in path-length based definitions, a node can be said to be more related to another node if there are short paths between them; however, unlike in path-based definitions, random walk-based definitions of relatedness also consider how tightly connected two nodes are and argue that nodes that have many paths between them can be considered more related.

Naturally, any distance measure which would require all paths among two nodes to be enumerated would require time exponential in the size of the graph and, thus, would be intractable. Thus, random-walk based techniques encode the structure of the network in the form a transition matrix of a stochastic process from which the node relationships can be inferred. When it exists, the convergence probability of a node n gives the ratio of the time spent at that node in a sufficiently long random walk and, therefore, neatly captures the connectivity of the node n in the graph. Therefore, many web search and recommendation algorithms, such as HITS (Kleinberg, 1999) and PageRank (Brin and Page, 1998), rely on random-walks to identify significant nodes in the graph. The well-known PageRank algorithm associates a single importance score to each node: Let us consider a weighted, directed graph $G(V, E)$, where the weight of the edge $e_j \in E$ is denoted as $w_j (\geq 0)$ and where

$$\left(\sum_{e_j \in outedge(v_i)} w_j \right) = 1.0.$$

The PageRank score of the node $v_i \in V$ is the stationary distribution of a random walk on G, where at each step

- with probability $1 - \beta$, the random walk moves along an outgoing edge of the current node with a probability proportional to the edge weights and
- with probability β, the walk jumps to a random node in V.

In other words, if we denote all the PageRank scores of the nodes in V with a vector $\vec{\pi}$, then

$$\vec{\pi} = (1 - \beta)\mathbf{T}_G \times \vec{\pi} + \beta\vec{j},$$

where $\mathbf{T}_G$ denotes the transition matrix corresponding to the graph G (and the underlying edge weights) and $\vec{j}$ is a *teleportation* vector where all entries are $\frac{1}{\|V\|}$.

1.1. Measuring Proximity with PageRank

The basic definition of PageRank associates a convergence score to each node in the graph irrespective of content and context of search. An early attempt to contextualize the PageRank scores was the *topic sensitive PageRank* (Haveliwala, 2002) approach which adjusts the PageRank scores of the nodes by assigning the *teleportation* probabilities in vector $\vec{j}$ in a way that reflects the graph nodes' degrees of match to the search topic. Candan and Li (2000) and Candan and Li (2002) were among the first works

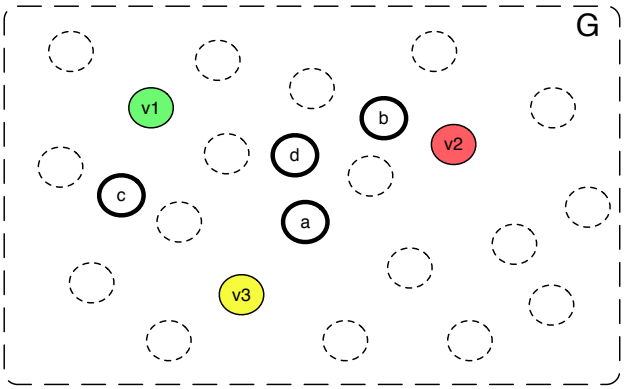


Fig. 1. Key questions: Given a graph, G, and a seed set of nodes $S = \{v_1, v_2, v_3\}$ in G, can we rank the remaining nodes in the graph regarding their relationships to the set S? Which of the nodes a through d is the most interesting given the seed set of nodes v_1 through v_3?

which recognized that random-walks can also be used for measuring the degree of *association*, *relatedness*, or *proximity* of the graph nodes to a given *seed node set*, $S \subseteq V$ (Figure 1): More specifically, Candan and Li (2000) construct a transaction matrix, $\mathbf{T}_S$, where edges leading away from the seed nodes are weighted less than those edges leading towards the seed nodes. Consequently, a page with a high connectivity but separated from the seed nodes in too many hops may be less significant within the context of seed nodes. In fact, a node which both has a high connectivity and few hops away from the seed nodes will have the highest convergence score. Therefore, when using the transition the convergence probabilities of the nodes capture both (a) the separations between the seeds and the graph nodes and (b) the connectivity of the nodes in the graph relative to nodes in S.

An alternative to this approach is to modify the teleportation vector, $\vec{j}$, as in topic sensitive PageRank (Haveliwala, 2002): instead of jumping to a random node in V with probability β, the random walk jumps to one of the nodes in the seed set, S, given by the user. More specifically, if we denote the *Personalized PageRank* scores of the nodes in V with a vector $\vec{\phi}$, then

$$\vec{\phi} = (1-\beta)\mathbf{T}_G \times \vec{\phi} + \beta\vec{s},$$

where $\vec{s}$ is a re-seeding vector, such that if $v_i \in S$, then $\vec{s}[i] = \frac{1}{\|S\|}$ and $\vec{s}[i] = 0$, otherwise. Intuitively, since at each step the random-walk has a non-zero probability of jumping back to the seed nodes from its current node (independently of where the current node is in the graph), the nodes closer to the nodes in S will have larger stationary scores than they would have if the random walk jumped randomly in the entire graph. One key advantage of this teleportation vector modification based approach over modifying the transition matrix, as in (Candan and Li, 2000), is that the term β can be used to directly control the *degree of seeding (or personalization)* of the PPR score. However, the use of personalized PageRank for large graphs is difficult due to the high cost of solving for the vector $\vec{\phi}$, given β, transition matrix $\mathbf{T}_G$, and the seeding vector $\vec{s}$.

One way to obtain $\vec{\phi}$ is to rewrite the stationary state equation of personalized

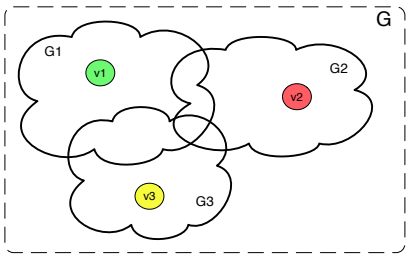


Fig. 2. Locality-sensitivity: Computation of PPR should focus on the neighborhoods (localities) of the seeds

PageRank as

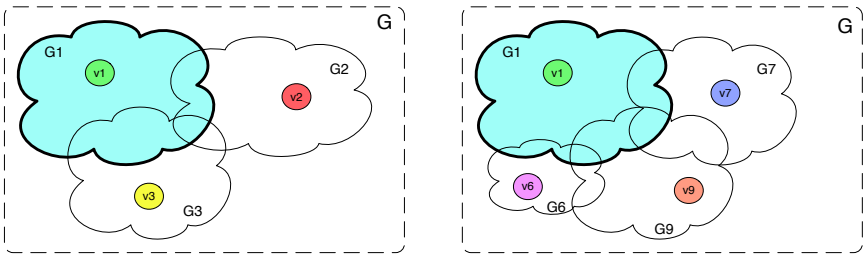
$$(\mathbf{I} - (1-\beta)\mathbf{T}_G) \times \vec{\phi} = \beta\vec{s},$$

and solve the above equation for $\vec{\phi}$ mathematically. Alternatively, PowerIteration methods (Kamvar et al, 2003) simulate the dissemination of probability mass by repeatedly applying the transition process to an initial distribution $\vec{\phi}_0$ until a convergence criterion is satisfied.

Unfortunately, for large data sets, both of these processes are prohibitively expensive. Recent advances on personalized PageRank include top-k and approximate personalized PageRank algorithms (Avrachenkov et al, 1995; Bahmani et al, 2010; Csalogany et al, 2005; Chakrabarti, 2007; Fujiwara et al, 2012; Gupta et al, 2008; Tong et al, 2006b; Song et al, 2009), parallelized implementations on MapReduce or Pregel based batch data processing systems (Bahmani et al, 2011; Malewicz et al, 2010), and techniques to eliminate seed noise (Huang et al, 2014). The FastRWR algorithm, presented in (Tong et al, 2006b), for example partitions the graph into subgraphs and indexes partial intermediary solutions. Given a seed node set S then relevant intermediary solutions are combined to quickly solve for approximate PPR scores. Naturally, there is a trade-off between the number of partitions created for the input graph G and the accuracy: the higher the number of partitions, the faster the run-time execution (and smaller the memory requirement), but higher the drop in accuracy. Unfortunately, as we see in Section 5, for large data sets, FastRWR requires large number of partitions to ensure that the intermediary metadata (which require dense matrix representation) fit into the available memory and this negatively impacts execution time and accuracy. (Maehara et al, 2014) proposes a block-based GMRES-PPR technique to speed up online computation of personalized PageRank. Experiments presented in Section 5 showed that while GMRES-PPRs online running time can be competitive, it can fail to complete pre-processing for large graphs as it runs out of memory – even when using a significantly better hardware setup than used for the proposed techniques. Moreover, the accuracy of GMRES-PPR is lower than that of the proposed techniques.

1.2. Contributions of this Paper

In this paper, we note that we can improve *both* scalability and accuracy through a *Locality-sensitive*, *Re-use promoting*, *approximate personalized PageRank* (LR-PPR) algorithm for efficiently computing the PPR values relying on the *localities* of the seed nodes on the graph: The LR-PPR algorithm is



(a) PPR query 1 (b) PPR query 2

Fig. 3. Re-use promotion: Two PPR queries sharing a seed node (v_1) should also share relevant work

- *locality sensitive* in the sense that it reduces the computational cost of the PPR computation process and improves accuracy by focusing on the neighborhoods of the seed nodes (Figure 2); and
- *re-use promoting* in that, instead of performing a monolithic PPR computation for a given seed node set S (where the intermediary results cannot be re-used for a different, but similar, seed set S'), LR-PPR divides the work into localities of the seeds and enables *caching* and *re-use* of significant portions of the intermediary work for the individual seed nodes in future queries (Figure 3): In other words, LR-PPR is able to leverage temporal locality in the users queries:
  - This temporal locality may be due to a slow evolution of a given users interest: for example when a user watches a new online movie, this will change the recommendation context only slightly as the users recent movie history (say the last 10 movies watched by the user) will be mostly preserved in the seed set.
  - This temporal locality may also be due to popular seeds shared by a lot of users: for example a new hit movie (or say the top 10 movies of the week) may be shared in the seed set of a large portion of the users. LR-PPR leverages such temporal localities to reduce redundant work.

In the following section, we first formally introduce the problem and then present our solution for *locality-sensitive*, *re-use promoting*, *approximate personalized PageRank* computations. In Section 4, we discuss optimization and parallelization opportunities. We evaluate LR-PPR for different data sets and under different scenarios in Section 5. We conclude in Section 6.

2. Proposed Approach

Let $G = (V, E)$ be a directed graph. For the simplicity of the discussion, without any loss of generality, let us assume that G is unweighted[1]. Let us be given a set $S \subseteq V$ of seed nodes (Figure 1) and a personalization parameter, β. Let $\mathcal{G}_S = \{G_h(V_h, E_h) \mid 1 \leq h \leq K\}$ be $K = \|S\|$ subgraphs[2] of G, such that

- for each $v_i \in S$, there exists a corresponding $G_i \in \mathcal{G}_S$ such that $v_i \in V_i$ and
- for all $G_h \in \mathcal{G}_S$, $\|G_h\| \ll \|G\|$.

[1] Extending the proposed algorithms to weighted graphs is trivial.
[2] We discuss alternative ways to select these in Section 4.

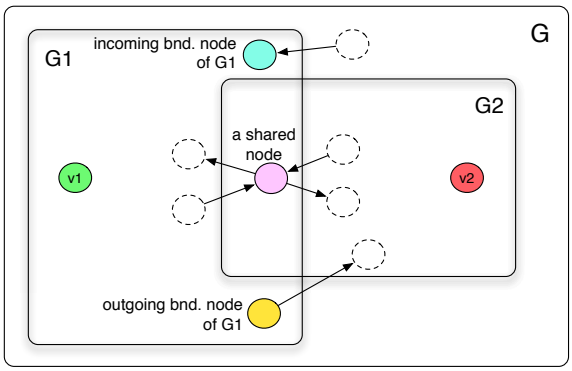


Fig. 4. Incoming and outgoing boundary nodes/edges and a node shared between two localities

We first formalize the locality-sensitivity goal (Figure 2):

Desideratum 1: Locality-Sensitivity. Our goal is to compute an approximate PPR vector, $\vec{\phi}_{apx}$, using $\mathcal{G}_S$ instead of G, such that $\vec{\phi}_{apx} \sim \vec{\phi}$, where $\vec{\phi}$ represents the true PPR scores of the nodes in V relative to S: i.e.,

$$\vec{\phi}_{apx} \sim \vec{\phi} = (1-\beta)\mathbf{T}_G \times \vec{\phi} + \beta\vec{s},$$

where $\mathbf{T}_G$ is the transition matrix corresponding to G and $\vec{s}$ is the re-seeding vector corresponding to the seed nodes in S.

We next formalize the re-use promotion goal (Figure 3):

Desideratum 2: Reuse-Promotion. Let S_1 and S_2 be two sets of seed nodes and let v_i be a node such that $v_i \in S_1 \cap S_2$. Let also the approximate PPR vector, $\vec{\phi}_{apx,1}$ corresponding to S_1 have already been computed using $\mathcal{G}_{S_1}$ and let us assume that the approximate PPR vector, $\vec{\phi}_{apx,2}$ corresponding to S_2 is being requested. The part of the work performed when processing $G_i \in \mathcal{G}_{S_1}$ (corresponding to v_i) should not need to be re-performed when processing $G_i \in \mathcal{G}_{S_2}$, when computing $\vec{\phi}_{apx,2}$ using $\mathcal{G}_{S_2}$.

2.1. Combined Locality and its Boundary

Unlike existing approximate PPR algorithms (Avrachenkov et al, 1995; Bahmani et al, 2010; Csalogany et al, 2005; Chakrabarti, 2007; Fujiwara et al, 2012; Gupta et al, 2008; Tong et al, 2006b; Song et al, 2009), LR-PPR is *location sensitive*. Therefore, given the set, S, of seed nodes and the corresponding localities, $\mathcal{G}_S$, the computation focuses on the *combined locality* $G^+(V^+, E^+) \subseteq G$, where

$$V^+ = \bigcup_{1 \le l \le K} V_l \text{ and } E^+ = \bigcup_{1 \le l \le K} E_l.$$

Given a combined locality, G^+, we can also define its *external graph*, $G^-(V^-, E^-)$, as the set of nodes and edges of G that are outside of G^+ and boundary nodes and edges. As shown in Figure 4, we refer to $v_i \in V_l$ as an *outgoing boundary node* of G_l if there

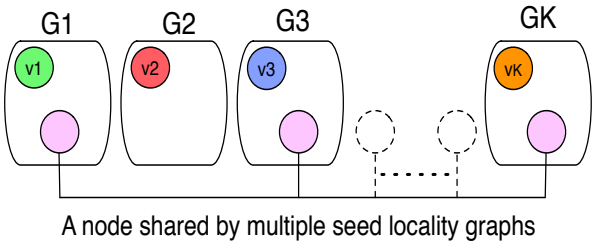


Fig. 5. An equivalence set consists of the copies of a node shared across multiple seed locality graphs

is an outgoing edge $e_{i,j} = [v_i \rightarrow v_j] \in E$, where $v_j \notin V_l$; the edge e_j is also referred to as an *outgoing boundary edge* of G_l. The set of all outgoing boundary nodes of G_l is denoted as $V_{outbound,l}$ and the set of all outgoing boundary edges of G_l is denoted as $E_{outbound,l}$. Note that $V_{outbound,l} \subseteq V_l$, whereas $E_{outbound,l} \cap E_l = \emptyset$.

We also define *incoming boundary nodes* ($V_{inbound,l}$) and *incoming boundary edges* ($E_{inbound,l}$) similarly to the outgoing boundary nodes and edges of G_l, but considering inbound edges to these subgraphs. More specifically, $E_{inbound,l}$ consists of edges of the form $[v_i \rightarrow v_j] \in E$, where $v_j \in V_l$ and $v_i \notin V_l$.

2.2. Localized Transition Matrix

Since LR-PPR focuses on the combined locality, G^+, the next step is to combine the transition matrices of the individual localities into a combined transition matrix. To produce accurate approximations, this *localized transition matrix* should nevertheless take the external graph, G^-, and the boundaries between G^- and G^+, into account.

2.2.1. Transition Matrices of Individual Localities

Let $v_{(l,i)}$ ($1 \leq l \leq K$) denote a re-indexing of vertices in V_l. If $v_{(l,i)} \in V_l$ and $v_c \in V$ s.t. $v_{(l,i)} = v_c$, we say that $v_{(l,i)}$ is a member of an *equivalence set*, $\mathcal{V}_c$ (Figure 5). Intuitively, the equivalence sets capture the common parts across the localities of the individual seed nodes. Given $G_l(V_l, E_l) \subseteq G$ and an appropriate re-indexing, we define the corresponding local transition matrix, $\mathbf{M}_l$, as a $\|V_l\| \times \|V_l\|$ matrix, where

- $\left(\nexists e_{i,j} = [v_{(l,i)} \rightarrow v_{(l,j)}] \in E_l\right) \rightarrow \mathbf{M}_l[j,i] = 0$ and
- $\left(\exists e_{i,j} = [v_{(l,i)} \rightarrow v_{(l,j)}] \in E_l\right) \rightarrow \mathbf{M}_l[j,i] = \frac{1}{out(v_{(l,i)})}$, where $out(v_{(l,i)})$ is the number of outgoing edges of v_i.

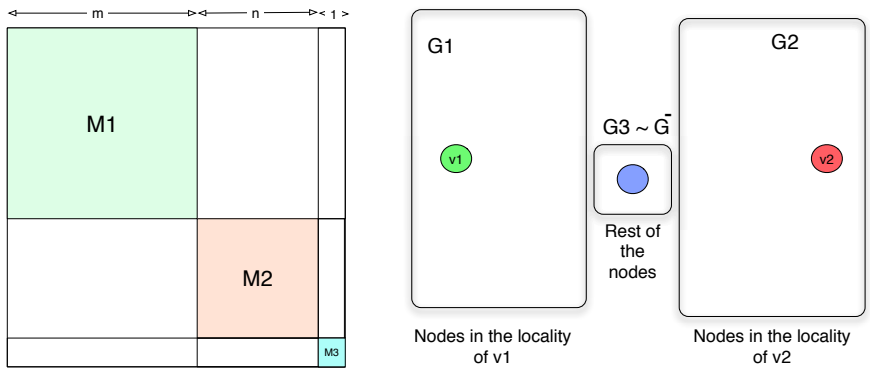
The $m \times m$ matrix $\mathbf{M}_2$ is also defined similarly considering edges in E_2.

2.2.2. Localization of the Transition Matrix

Given the local transition matrices, $\mathbf{M}_1$ through $\mathbf{M}_K$, we *localize* the transition matrix of G by approximating it as

$$\mathbf{M}_{apx} = \mathbf{M}_{bd} + \mathbf{M}_0,$$

where $\mathbf{M}_{bd}$ is a block-diagonal matrix of the form



(a) $\mathbf{M}_{bd}$ matrix (b) corresponding graph

Fig. 6. The matrix, $\mathbf{M}_{bd}$, (a) ignores the overlaps among the localities of the seed nodes, (b) ignores the outgoing and incoming edges from/to the localities and collapses all external nodes/edges to a single node

$$\begin{pmatrix} \mathbf{M}_1 & \mathbf{0}_{\|V_1\|\times\|V_2\|} & \dots & \mathbf{0}_{\|V_1\|\times\|V_K\|} & \mathbf{0}_{\|V_1\|\times 1} \\ \mathbf{0}_{\|V_2\|\times\|V_1\|} & \mathbf{M}_2 & \dots & \mathbf{0}_{\|V_2\|\times\|V_K\|} & \mathbf{0}_{\|V_2\|\times 1} \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0}_{\|V_K\|\times\|V_1\|} & \mathbf{0}_{\|V_K\|\times\|V_2\|} & \dots & \mathbf{M}_K & \mathbf{0}_{\|V_K\|\times 1} \\ \mathbf{0}_{1\times\|V_1\|} & \mathbf{0}_{1\times\|V_2\|} & \dots & \mathbf{0}_{1\times\|V_K\|} & \mathbf{M}_{K+1} \end{pmatrix},$$

where $\mathbf{M}_{K+1}$ is equal to the 1×1 matrix $\mathbf{0}_{1\times 1}$. Intuitively, $\mathbf{M}_{bd}$ combines the K subgraphs into one transition matrix, without considering common nodes/edges or incoming/outgoing boundary edges and ignoring all outgoing and incoming edges (Figure 6). All the external nodes in G^- are accounted by a single node represented by the 1×1 matrix $\mathbf{M}_{K+1}$.

As we see later in Section 3, a key advantage of $\mathbf{M}_{bd}$ is that it is block-diagonal and, hence, there are efficient ways to process it. However, this block-diagonal matrix, $\mathbf{M}_{bd}$, cannot accurately represent the graph G as it ignores potential overlaps among the individual localities and ignores all the nodes and edges outside of G^+. We therefore need a *compensation matrix* to

- make sure that nodes and edges shared between the localities are not double counted during PPR computation and
- take into account the topology of the graph external to localities G_1 through G_K.

2.2.3. *Compensation Matrix,* $\mathbf{M}_0$

Let t be $(\|V_1\|+\|V_2\|+\ldots+\|V_K\|+1)$. The compensation matrix, $\mathbf{M}_0$, is a $t \times t$ matrix accounting for the boundary edges of the seed localities as well as the nodes/edges in G^-. $\mathbf{M}_0$ also ensures that the common nodes in V_1 through V_K are not double counted during PPR calculations. $\mathbf{M}_0$ is constructed as follows:

Row/column indexing: Let $v_{l,i}$ be a vertex in V_l. We introduce a row/column indexing

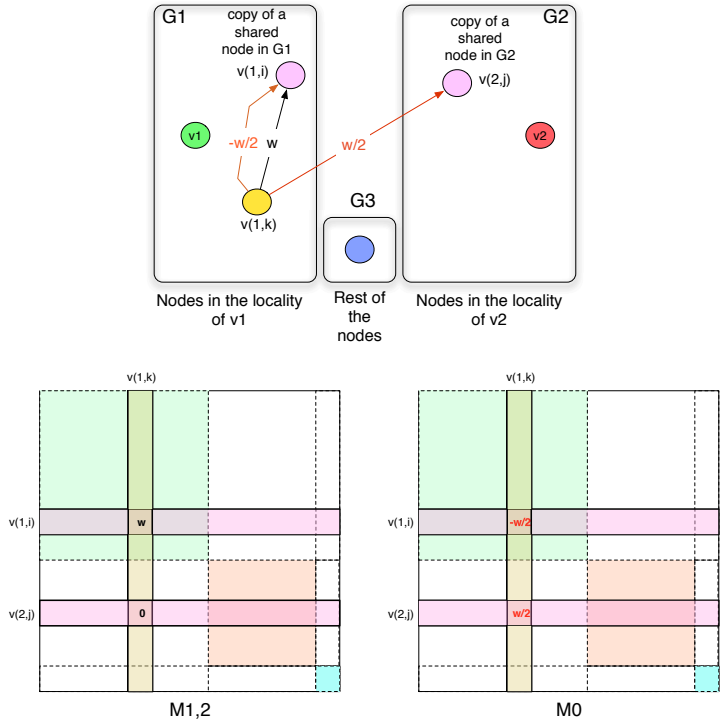


Fig. 7. Accounting for shared nodes in the compensation matrix, $\mathbf{M}_0$: in this example, half of the transitions are re-routed to the copy of the node (note that, $w = \frac{1}{out(G, v_{1,k})}$)

function, $ind()$, defined as follows:

$$ind(l, i) = \left(\sum_{1 \leq h < l} \|V_h\| \right) + i$$

Intuitively the indexing function, $ind()$, maps the relevant nodes in the graph to their positions in the $\mathbf{M}_0$ matrix.

Compensation for the common nodes: Let $e_{l,i,j}$ be an edge $[v_{(l,i)} \rightarrow v_{(l,j)}] \in E_l$ and let $v_{(l,j)}$ be a member of the equivalence set $\mathcal{V}_c$ for some $v_c \in V$. Then, if $\|\mathcal{V}_c\| > 1$

– $\mathbf{M}_0[ind(l,j), ind(l,i)] = -(\frac{1}{out(G_l, v_{l,i})} - \frac{\|\mathcal{V}_c\|-1}{\|\mathcal{V}_c\|} \times \frac{1}{out(G, v_{l,i})})$ and

– $\forall v_{(h,k)} \in \mathcal{V}_c$ s.t. $v_{(h,k)} \neq v_{(l,j)}$, we have

$$\mathbf{M}_0[ind(h,k), ind(l,i)] = \frac{1}{\|\mathcal{V}_c\|} \times \frac{1}{out(G, v_{l,i})},$$

where $out(G, v)$ is the outdegree of node v in G and $out(G_l, v)$ is the outdegree of node v in the subgraph G_l. Intuitively, the compensation matrix re-routes a portion of the transitions going towards a shared node in a given locality V_l to the copies in other seed localities (Figure 7). This prevents the transitions to and from the shared node from being mis-counted.

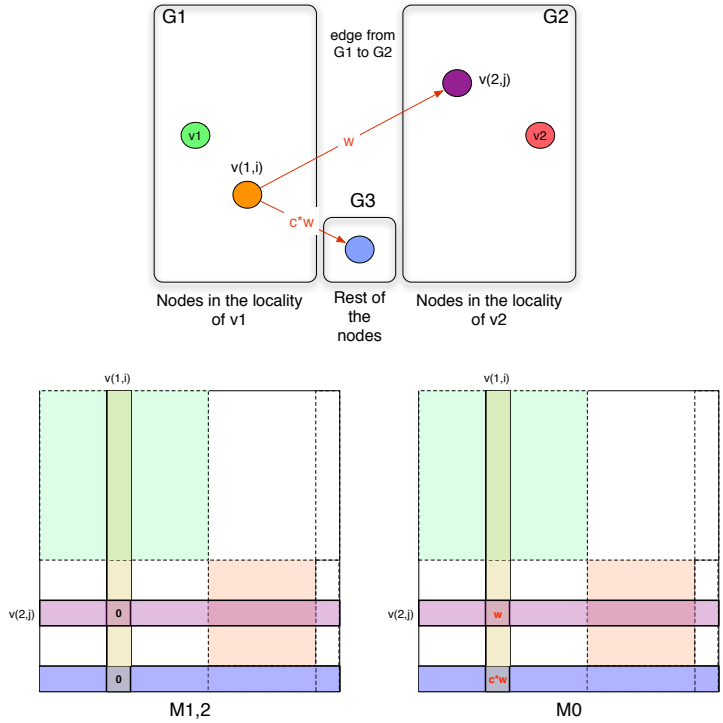


Fig. 8. Accounting for the edges that are outgoing from a locality (in this example, $w = \frac{1}{out(v_{1,i})}$ and $c = bnd(v_{1,i})$)

Compensation for outgoing boundary edges: The compensation matrix needs to account also for outgoing boundary edges that are not accounted for by the neighborhood transition matrices $\mathbf{M}_1$ through $\mathbf{M}_K$:

– *Accounting for boundary edges from nodes in V_l to nodes in V_h*: $\forall[v_{(l,i)} \rightarrow v_{(h,j)}] \in E_{outbound,l}$
  - $\mathbf{M}_0[ind(h,j), ind(l,i)] = \frac{1}{out(v_{(l,i)})}$ and
  - $\mathbf{M}_0[ind(l,p), ind(l,i)] = -(\frac{1}{out(G_k, v_{l,i})} - \frac{1}{out(G, v_{(l,i)})})$, where $\exists e_{i,p} = [v_{(l,i)} \rightarrow v_{(l,p)}] \in E_l$ and $v_{l,p}$ is not a member of the equivalence set $\mathcal{V}_c$ for any $v_c \in V$

– *Accounting for boundary edges from nodes in V_l to graph nodes that are in V^-*: if $\exists[v_{(l,i)} \rightarrow v] \in E_{outbound,l}$ s.t. $v \in V^-$
  - $\mathbf{M}_0[t, ind(l,i)] = \frac{bnd(v_{(l,i)})}{out(v_{(l,i)})}$, where $bnd(v_{(l,i)})$ is the number of edges of the form $[v_{(l,i)} \rightarrow v] \in E_{outbound,l}$ where $v \in V^-$

  else $\mathbf{M}_0[t, ind(l,i)] = 0$

The process of compensating for outgoing boundary edges for a sample case when $K = 2$ is visualized in Figure 8. The compensation matrix records all outgoing edges,

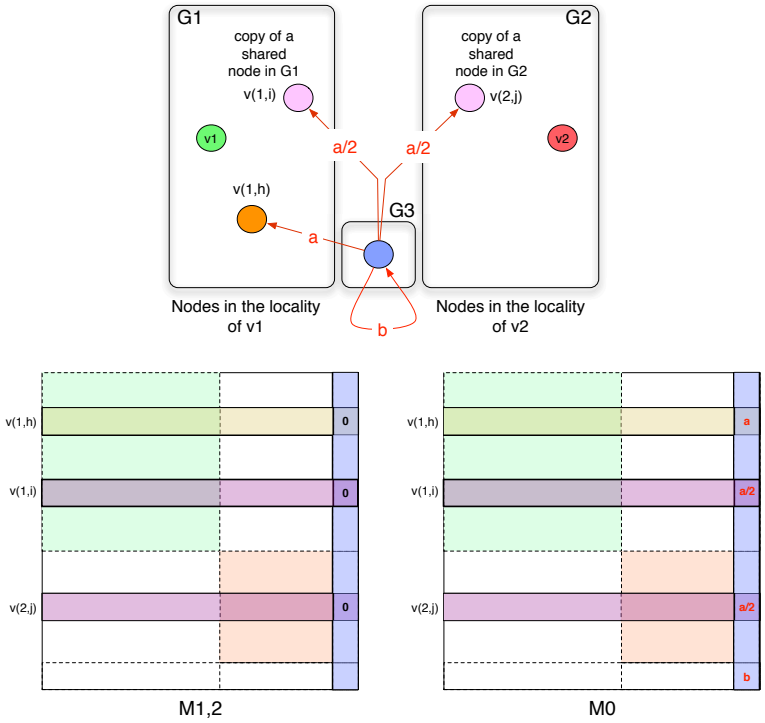


Fig. 9. Accounting for the edges that are originating from the nodes that are outside of the localities of G_1 and G_2

whether they cross into another locality or they are into external nodes in G^-. If a node has more than one outgoing edge into the nodes in G^-, all such edges are captured using one single compensation edge which aggregates all the corresponding transition probabilities.

Compensation for incoming boundary edges (from G^-): Similarly to the outgoing boundary edges, the compensation matrix needs also to account for incoming boundary edges that are not accounted for by the neighborhood transition matrices $\mathbf{M}_1$ through $\mathbf{M}_K$. Since incoming edges from other localities have been accounted for in the previous step, here we only need to consider incoming boundary edges (from G^-). Following the formulation in Wu and Raschid (2009), we account for incoming edges where the source is external to G^+ and the destination is a vertex $v_{(l,i)}$ in V_l by inserting an edge from the dummy node to $v_{(l,i)}$ with a weight that considers the outdegrees of all external source nodes; i.e., $\forall v_{(l,i)}$ s.t. $\exists[v_k \rightarrow v_{(l,i)}] \in E_{inbound,l}$ where $v_k \in V^-$ and $v_{(l,i)}$ is in the equivalence set $\mathcal{V}_c$ for a $v_c \in V$, $\mathbf{M}_0[ind(l,i),t]$ is equal to

$$\frac{1}{\|\mathcal{V}_c\|} \frac{\sum_{([v_k \rightarrow v_{(l,i)}] \in E_{inbound,l}) \wedge (v_k \in V^-)} \frac{1}{out(G,v_k)}}{\|V^-\|},$$

where $out(G,v)$ is the outdegree of node v in G. The process of compensating for

incoming edges originating from outside of the locality graphs of the seeds is visualized in Figure 9.

Compensation for the edges in G^-: We account for edges that are entirely in G^- by creating a self-loop that represents the sum of outdegree flow between all external nodes averaged by the number of external nodes; i.e.,

$$\mathbf{M}_0[t,t] = \frac{\sum_{v \in V^-} \frac{out(G^-,v)}{out(G,v)}}{\|V^-\|},$$

where $out(G^-, v)$ and $out(G, v)$ are the outdegrees of node v in G^- and G, respectively. The process of compensating for edges that are outside of the seed localities is also visualized in Figure 9.

Completion: For any matrix position p, q not considered above, no compensation is necessary; i.e., $\mathbf{M}_0[p, q] = 0$.

2.3. L-PPR: Locality Sensitive PPR

Once the block-diagonal local transition matrix, $\mathbf{M}_{bd}$, and the compensation matrix, $\mathbf{M}_0$, are obtained, the next step is to obtain the PPR scores of the nodes in V^+. This can be performed using any fast PPR computation algorithm discussed in Section 1.1.

Note that the overall transition matrix $\mathbf{M}_{apx} = \mathbf{M}_{bd} + \mathbf{M}_0$ is approximate in the sense that all the nodes external to G^+ are clustered into a single node, represented by the last row and column of the matrix. Otherwise, the combined matrix $\mathbf{M}_{apx}$ accurately represents the nodes and edges in the "*merged localities graph*" combining the seed localities, G_1 through G_K. As we see in Section 5, this leads to highly accurate PPR scores with better scalability than existing techniques.

2.4. LR-PPR: Locality Sensitive and Reuse Promoting PPR

Our goal is not only to leverage locality-sensitivity as in L-PPR, but also to boost sub-result re-use. Let us restate the problem: Given the block-diagonal local transition matrix, $\mathbf{M}_{bd}$, and the compensation matrix, $\mathbf{M}_0$ (that together make up the overall transition matrix, $\mathbf{M}_{axp}$) computed as described above, and a re-seeding (or restart) probability, β, we seek to find $\vec{\phi}_{apx}$, where

$$\vec{\phi}_{apx} = \beta(\mathbf{I} - (1-\beta)\mathbf{M}_{axp})^{-1}\vec{s},$$

where $\vec{s}$ is the re-seeding vector for seeds. Remember that, as discussed above, the localized transition matrix $\mathbf{M}_{apx}$ is equal to $\mathbf{M}_{bd} + \mathbf{M}_0$ where (by construction) $\mathbf{M}_{bd}$ is a block-diagonal matrix, whereas $\mathbf{M}_0$ (which accounts for shared, boundary, and external nodes) is relatively sparse. We next use these two properties of the decomposition of $\mathbf{M}_{apx}$ to efficiently compute approximate PPR scores of the nodes in V^+. In particu-

lar, we rely on the following result due to Tong et al (2006b), which itself relies on the Sherman-Morisson lemma Piegorsch and Casella (1990):

> Let $\mathbf{C} = \mathbf{A} + \mathsf{U}\mathbf{S}\mathbf{V}$. Let also $(\mathbf{I} - c\mathbf{A})^{-1} = \mathbf{Q}^{-1}$. Then, the equation
>
> $$\vec{r} = (1-c)(\mathbf{I} - c\mathbf{A})^{-1}\vec{e}$$
>
> has the solution
>
> $$\vec{r} = (1-c)(\mathbf{Q}^{-1}\vec{e} + c\mathbf{Q}^{-1}\mathbf{U}\mathbf{\Lambda}\mathbf{V}\mathbf{Q}^{-1}\vec{e}),$$
>
> where
>
> $$\mathbf{\Lambda} = (\mathbf{S}^{-1} - c\mathbf{V}\mathbf{Q}^{-1}\mathbf{U})^{-1}.$$
>
> If $\mathbf{A}$ is a block diagonal matrix consisting of k blocks, $\mathbf{A}_1$ through $\mathbf{A}_k$, then $\mathbf{Q}^{-1}$ is also a block diagonal matrix consisting of k corresponding blocks, $\mathbf{Q}_1^{-1}$ through $\mathbf{Q}_k^{-1}$, where $\mathbf{Q}_i^{-1} = (\mathbf{I} - c\mathbf{A}_i)^{-1}$.

We use the above observation to efficiently obtain PPR scores by setting $c = (1-\beta)$, $\mathbf{C} = \mathbf{M}_{apx}$, $\mathbf{A} = \mathbf{M}_{bd}$, and $\mathsf{U}\mathbf{S}\mathbf{V} = \mathbf{M}_0$. In particular, we divide the PPR computation into two steps: a locality-sensitive and re-usable step involving the computation of the $\mathbf{Q}^{-1}$ term using the local transition matrices and a run-time computation step involving the compensation matrix.

2.4.1. Locality-sensitive and Re-usable $\mathbf{Q}_{bd}^{-1}$

Local transition matrices, $\mathbf{M}_1$ through $\mathbf{M}_K$ corresponding to the seeds v_1 through v_K are constant (unless the graph itself evolves over time). Therefore, if $\mathbf{Q}_h^{-1} = (\mathbf{I} - (1-\beta)\mathbf{M}_h)^{-1}$ is computed and cached once, it can be reused for obtaining $\mathbf{Q}_{bd}^{-1}$, which is a block diagonal matrix consisting of $\mathbf{Q}_1^{-1}$ through $\mathbf{Q}_{K+1}^{-1}$ (as before, the last block, $\mathbf{Q}_{K+1}^{-1}$, is simply equal to $1_{1\times 1}$):

$$\begin{pmatrix} \mathbf{Q}_1^{-1} & \mathbf{0}_{\|V_1\|\times\|V_2\|} & \cdots & \mathbf{0}_{\|V_1\|\times\|V_K\|} & \mathbf{0}_{\|V_1\|\times 1} \\ \mathbf{0}_{\|V_2\|\times\|V_1\|} & \mathbf{Q}_2^{-1} & \cdots & \mathbf{0}_{\|V_2\|\times\|V_K\|} & \mathbf{0}_{\|V_2\|\times 1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{0}_{\|V_K\|\times\|V_1\|} & \mathbf{0}_{\|V_K\|\times\|V_2\|} & \cdots & \mathbf{Q}_K^{-1} & \mathbf{0}_{\|V_K\|\times 1} \\ \mathbf{0}_{1\times\|V_1\|} & \mathbf{0}_{1\times\|V_2\|} & \cdots & \mathbf{0}_{1\times\|V_K\|} & \mathbf{Q}_{K+1}^{-1} \end{pmatrix},$$

2.4.2. Computation of the LR-PPR Scores

In order to be able to use the above formulation for obtaining the PPR scores of the nodes in V^+, in the query time, we need to decompose the compensation matrix, $\mathbf{M}_0$, into $\mathsf{U}_0\mathbf{S}_0\mathbf{V}_0$. While obtaining a precise decomposition in run-time would be prohibitively expensive, since $\mathbf{M}_0$ is sparse and since we are looking for an approximation of the PPR scores, we can obtain a fairly accurate low-rank approximation of $\mathbf{M}_0$ efficiently (Tong et al, 2006b):

$$\mathbf{M}_0 \simeq \tilde{\mathbf{U}}_0\tilde{\mathbf{S}}_0\tilde{\mathbf{V}}_0.$$

Given this decomposition, the result vector $\vec{\phi}_{apx}$, which contains the (approximate) PPR scores of the nodes in V^+, is computed as

$$\vec{\phi}_{apx} = \beta \left(\mathbf{Q}_{bd}^{-1}\vec{s} + (1-\beta)\mathbf{Q}_{bd}^{-1}\tilde{\mathbf{U}}_0\mathbf{\Lambda}\tilde{\mathbf{V}}_0\mathbf{Q}_{bd}^{-1}\vec{s} \right),$$

where

$$\mathbf{\Lambda} = \left(\tilde{\mathbf{S}}_0^{-1} - (1-\beta)\tilde{\mathbf{V}}_0\mathbf{Q}_{bd}^{-1}\tilde{\mathbf{U}}_0 \right)^{-1}.$$

Note that the compensation matrix $\mathbf{M}_0$ is query specific and, thus, the work done for the last step cannot be reused across queries. However, as we experimentally verify in Section 5, the last step is relatively cheap and the earlier (costlier) steps involve re-usable work. Thus, caching and re-use through LR-PPR enables significant savings in execution time. We discuss the overall complexity and the opportunities for re-use next.

2.5. Accuracy Analysis

In this section, we analyze the accuracy of the proposed LR-PPR technique. As an approximation based technique, LR-PPR has two sources of errors: (a) errors that arise due to the use of a single node to represent the external graph G^- and (b) errors due to the use of a low-rank approximation to obtain PPR scores. In this section, we will analyze errors due to each separately.

2.5.1. Accuracy Loss due to the Use of a Single Node to Represent G^-

As discussed in the previous section, the LR-PPR algorithm collapses the portion of the graph external to the neighborhoods of the selected seeds (i.e., G^-) into a single *dummy* node – which leads to significant savings in execution time.

On the other hand, this operation can also result in a certain loss of accuracy. This is because, while we are combining the set, V^-, of nodes outside of the seeds' locality graphs into a single node, we also need to associate transition weights from this combined dummy node into the nodes in the locality graphs (which we refer to as *inbound edges* in the previous section).

Let us denote the set of all inbound edges from G^- to G^+ as $E_{inbound}$. Let us also denote the subset of vertices in V^- that are sources of edges in $E_{inbound}$ as $V^-_{inbound}$. It is easy to see that if for all $e\langle v_i, v_j\rangle \in E_{inbound}$, the overall transition volume from v_i onto v_j in the original graph is identical to the overall transition volume from the dummy node, $v_\perp$, onto v_j in the compressed graph, then the overall stationary distributions for the nodes in V^+ would be preserved: this is because for the purposes of the nodes in V^+ all incoming and outgoing transitions would be identical before and after the transformation. Therefore, to estimate the error due to the transformation, we need to investigate the transition volume on the edges in $E_{inbound}$.

Let $e = \langle v_i, v_j\rangle \in E_{inbound}$ be an in-bound edge from $v_i \in V^-_{inbound}$ to $v_j \in V^+$. Similarly, let $e' = \langle v_\perp, v_j\rangle$ be the corresponding edge in the *compressed* graph, where $v_\perp$ is the dummy node:

- *Volume of transitions on e*: On the original graph, the total volume, $vol(e)$, of transitions from v_i onto v_j can be computed as

$$vol(e) = P(v_i) \times \frac{1}{outdeg(v_i)}.$$

– *Volume of transitions on* e': On the compressed graph, the total volume, $vol(e')$, of transitions from $v_\perp$ onto v_j, on the other hand, can be computed as

$$vol(e') = P(v_\perp) \times \omega(e'),$$

where we use $\omega(e')$ to denote the transition weight associated to e', computed by accounting the outdegrees of all external source nodes as described in the previous section.

Consequently the transformation is lossless if and only if $\omega(e')$ is such that

$$\omega(e') = \frac{P(v_i)}{P(v_\perp)} \times \frac{1}{outdeg(v_i)},$$

which would also imply that

$$\omega(e') = \frac{P(v_i)}{\sum_{v \in V^-} P(v)} \times \frac{1}{outdeg(v_i)}.$$

This implies that the value $\omega(e')$ which would lead to a lossless transformation depends on the stationary probability distribution of v_i as well as the stationary probability distribution of $v \in V^-$ on the original graph. Unfortunately, these distributions are not available ahead of the time – therefore this value cannot be set in a lossless way *a priori*.

Wu and Raschid (2009) provide an analysis of the L_1 error between the PageRank scores with and without the use of a dummy node and the authors show that the L_1 error of the PageRank scores of the whole graph is within a constant factor of the L_1 error of the PageRank scores of the external nodes. Their formulation, however, does not explain under what conditions the overall error is low; moreover, the L_1 based formulation has the disadvantage that it does not provide a tight bound on the error for scores of individual nodes – it is possible to have a relatively low total L_1 error, which nevertheless may correspond to a large error on a specific node. In the next of the section, therefore, we aim to identify a tighter bound on the individual nodes, which also explains under what conditions the error is low.

Let $\mathbf{M}_{ideal}$ (or $\mathbf{M}$ for short) denote the $(|V^+|+1) \times (|V^+|+1)$ (or $N \times N$ for short) transition matrix based on the ideal transformation and $\mathbf{M}_{apx}$ (or $\mathbf{M}'$ for short) denote the transition matrix obtained as described in the previous section, without having a priori knowledge of the stationary distributions of the nodes in V^-. Let $\delta\mathbf{M} = \mathbf{M} - \mathbf{M}'$ denote the error matrix – note that $\delta\mathbf{M}$ matrix consists largely of zeros except for the very last column corresponding to outgoing transitions from the dummy node.

Remember that the personalized PageRank scores, $\vec{\pi}$, given transition matrix $\mathbf{M}$ can be written as

$$\vec{\pi} = (1-\beta)\mathbf{M}\vec{\pi} + \beta\vec{s},$$

where β is the re-seeding probability and $\vec{s}$ is the re-seeding vector. Similarly, we have

$$\vec{\pi}' = (1-\beta)\mathbf{M}'\vec{\pi}' + \beta\vec{s},$$

for the approximate transition matrix $\mathbf{M}'$.

Here we are interested to quantify $\delta\vec{\pi} = \vec{\pi} - \vec{\pi}'$ which gives the errors in the personalized PageRank scores due to the approximation. Let us, instead, consider the $\delta\vec{x} = \vec{x} - \vec{x}'$, where

$$\vec{x} = \mathbf{M}\vec{x} \quad \text{and} \quad \vec{x}' = \mathbf{M}'\vec{x}';$$

i.e., $\vec{x}$ and $\vec{x}'$ are the steady-state distributions of the random walks based on the transition matrices $\mathbf{M}$ and $\mathbf{M}'$, respectively. Since the seed nodes are, by construction, *away* from the dummy node, the steady state distributions of the dummy node in $\vec{x}$ are likely to be lower than that in $\vec{\pi}$. However, as we have seen above, the approximation error is due to the transitions originating from the dummy node into the nodes in V^+ – consequently, we expect that errors in $\delta\vec{x}$ will be larger than the errors in $\delta\vec{\pi}$. Therefore, instead of studying $\delta\vec{\pi}$ directly, we can use $\delta\vec{x}$ to gauge the scale of the error in personalized PageRank scores.

Note that $\vec{x}$ and $\vec{x}'$ are the eigenvectors with eigenvalue 1 of the transition matrices $\mathbf{M}$ and $\mathbf{M}'$, respectively. In general, we can formulate the eigenvalues and eigenvectors of $\mathbf{M}$ and $\mathbf{M}'$ as follows:

$$\mathbf{M}\vec{x}_h = \lambda_h \vec{x}_h$$

and

$$\mathbf{M}'\vec{x}'_h = \lambda'_h \vec{x}'_h,$$

where λ_h is the hth largest eigenvalue and $\vec{x}_h$ is the corresponding eigenvector. Since both $\mathbf{M}$ and $\mathbf{M}'$ correspond to random walk graphs, for both of them, the largest eigenvalue will be equal to 1. For convenience, let us assume the eigenvalues are ordered in non-increasing order, such that $\lambda_1 = \lambda'_1 = 1$.

Let us first take the equation $\mathbf{M}'\vec{x}'_h = \lambda'_h \vec{x}'_h$ and expand it with replacing $\mathbf{M}'$ with $\mathbf{M} + \delta\mathbf{M}$, and $\vec{x}'$ with $\vec{x} + \delta\vec{x}$, and λ' with $\lambda + \delta\lambda$:

$$\mathbf{M}\vec{x}_h + \delta\mathbf{M}\vec{x}_h + \mathbf{M}\delta\vec{x}_h + \delta\mathbf{M}\delta\vec{x}_h = \lambda_h\vec{x}_h + \lambda_h\delta\vec{x}_h + \delta\lambda_h\vec{x}_h + \delta\lambda_i\delta\vec{x}_h.$$

This becomes

$$\delta\mathbf{M}\vec{x}_h + \mathbf{M}\delta\vec{x}_h + \delta\mathbf{M}\delta\vec{x}_h = \lambda_h\delta\vec{x}_h + \delta\lambda_h\vec{x}_h + \delta\lambda_i\delta\vec{x}_h$$

after we cancel terms using $\mathbf{M}\vec{x}_h = \lambda_h\vec{x}_h$. To enable further simplifications, let us replace the term $\delta\vec{x}_h$ with $\sum_{l=1}^{R}\alpha_{hl}\vec{x}_l$, by mapping the error vector $\delta\vec{x}_h$ to the vector space defined by the R eigenvectors, $\vec{x}_l$, where $1 \leq l \leq R \leq N = |V^+| + 1$. Here R is the rank[3] of the transition matrix $\mathbf{M}$ and α_{hl} are constants that are to be determined. After we replace the term $\delta\vec{x}_h$ with its description in the eigenspace, we obtain

$$\delta\mathbf{M}\vec{x}_h + \mathbf{M}\sum_{l=1}^{R}\alpha_{hl}\vec{x}_l + \delta\mathbf{M}\delta\vec{x}_h = \lambda_h\sum_{l=1}^{R}\alpha_{hl}\vec{x}_l + \delta\lambda_h\vec{x}_h + \delta\lambda_h\delta\vec{x}_h.$$

Since, $\mathbf{M}\vec{x}_l = \lambda_l\vec{x}_l$, we can rewrite this as

$$\delta\mathbf{M}\vec{x}_h + \sum_{l=1}^{R}\alpha_{hl}\lambda_l\vec{x}_l + \delta\mathbf{M}\delta\vec{x}_h = \lambda_h\sum_{l=1}^{R}\alpha_{hl}\vec{x}_l + \delta\lambda_h\vec{x}_h + \delta\lambda_h\delta\vec{x}_h.$$

Assuming that the eigenvectors are normalized[4] to length 1 and given that they are mutually orthogonal, we can eliminate summations by left multiplying both sides of the above equation by $\vec{x}_k^\top$:

$$\vec{x}_k^\top\delta\mathbf{M}\vec{x}_h + \alpha_{hk}\lambda_k + \vec{x}_k^\top\delta\mathbf{M}\delta\vec{x}_h = \lambda_h\alpha_{hk} + \vec{x}_k^\top\delta\lambda_h\vec{x}_h + \vec{x}_k^\top\delta\lambda_h\delta\vec{x}_h.$$

[3] If the rank of $\mathbf{M}$ is lower than the rank of $\mathbf{M}'$, then we can formulate the error term $\delta\vec{x}_h$ in terms of the eigenvectors of $\mathbf{M}'$ – the rest of the discussion would be modified accordingly.

[4] This would simply scale the PageRank scores and their approximation errors proportionally.

Moreover, since $\vec{x}'_h = \vec{x}_h + \delta\vec{x}_h$,

$$\vec{x}_k^\top \delta\mathbf{M}\vec{x}'_h + \alpha_{hk}\lambda_k = \lambda_h\alpha_{hk} + \vec{x}_k^\top \delta\lambda_h\vec{x}'_h.$$

Rearranging this for α_{hk}, we get

$$\alpha_{hk} = \frac{\vec{x}_k^\top(\delta\mathbf{M} - \delta\lambda_h)\vec{x}'_h}{\lambda_h - \lambda_k}, \quad \text{for} \quad k \neq h.$$

This, however, does not give us the value of α_{hh}. To solve for α_{hh}, we rely on the fact that (as stated above) the eigenvectors are assumed normalized to be orthonormal:

$$(\vec{x}'_h)^\top \vec{x}'_h = (\vec{x}_h + \delta\vec{x}_h)^\top(\vec{x}_h + \delta\vec{x}_h) = (\vec{x}_h^\top + \delta\vec{x}_h^\top)(\vec{x}_h + \delta\vec{x}_h) = 1.$$

From this, we get

$$\vec{x}_h^\top\vec{x}_h + \delta\vec{x}_h^\top\vec{x}_h + \vec{x}_h^\top\delta\vec{x}_h + \delta\vec{x}_h^\top\delta\vec{x}_h = 1.$$

Because $\vec{x}_h^\top\vec{x}_h = 1$, we can further simplify this as

$$\delta\vec{x}_h^\top\vec{x}_h + \vec{x}_h^\top\delta\vec{x}_h + \delta\vec{x}_h^\top\delta\vec{x}_h = 0.$$

Here the term $\vec{x}_h^\top\delta\vec{x}_h$ can be further simplified into α_{hh} by considering the mapping of $\delta\vec{x}_h$ into the vector space defined by the orthonormal eigenvectors:

$$\vec{x}_h^\top\delta\vec{x}_h = \vec{x}_h^\top\sum_{l=1}^{R}\alpha_{hl}\vec{x}_l = \alpha_{hh}.$$

Similarly, we can simplify the term $\delta\vec{x}_h^\top\vec{x}_h$ into α_{hh} by noting that $\delta\vec{x}_h^\top\vec{x}_h$ is also equal to $\vec{x}_h^\top\delta\vec{x}_h$. Consequently, we obtain the equality

$$2\alpha_{hh} + \delta\vec{x}_h^\top\delta\vec{x}_h = 0.$$

which leads to the following term for α_{hh}:

$$\alpha_{hh} = -\frac{1}{2}\delta\vec{x}_h^\top\delta\vec{x}_h.$$

Having solved for α_{hk} for all values of h and k, we are now ready to replace the α terms in the eigenspace representation of $\delta\vec{x}_h$,

$$\delta\vec{x}_h = \alpha_{hh}\vec{x}_h + \sum_{\substack{l=1\\ l\neq h}}^{R}\alpha_{hl}\vec{x}_l,$$

with the corresponding α values:

$$\delta\vec{x}_h = \left(-\frac{1}{2}\delta\vec{x}_h^\top\delta\vec{x}_h\right)\vec{x}_h + \sum_{\substack{l=1\\ l\neq h}}^{R}\left(\frac{\vec{x}_l^\top(\delta\mathbf{M} - \delta\lambda_h)\vec{x}'_h}{\lambda_h - \lambda_l}\right)\vec{x}_l.$$

Note that this enables us quantify the error in all eigenvectors of the transition matrix. However, we are interested only in the amount of error in the eigenvector corresponding to the eigenvalue equal to 1:

$$\delta\vec{x} = \delta\vec{x}_1 = \left(-\frac{1}{2}\delta\vec{x}_1^\top\delta\vec{x}_1\right)\vec{x}_1 + \left(\sum_{l=2}^{R}\frac{\vec{x}_l^\top\delta\mathbf{M}\vec{x}'_1}{1-\lambda_l}\vec{x}_l\right).$$

Note that the term $\delta\lambda_1$ disappeared as, when $h = 1$, both $\mathbf{M}$ and $\mathbf{M}'$ have their largest eigenvalue equal to 1; i.e., $\delta\lambda_1 = 1 - 1 = 0$. Moreover, since the term $\delta\vec{x}_1^\top \delta\vec{x}_1$ is the square of the L_2 norm of the error vector (which is the difference of two vectors, each normalized to length 1), $\delta\vec{x}$ will be dominated by the second term:

$$\delta\vec{x} \sim \sum_{l=2}^{R} \frac{\vec{x}_l^\top \delta\mathbf{M}\vec{x}'}{1-\lambda_l}\vec{x}_l = \sum_{l=2}^{R} \Theta_l \vec{x}_l$$

From this formulation of the error vector, $\delta\vec{x}$, we can deduce the following: First of all, the magnitude of Θ_l (and consequently that of the error) depends on the separation of the largest eigenvalue (i.e., 1) and the other eigenvalues of the transition matrix $\mathbf{M}$. In fact, if for any $l > 1$, the eigenvalue is very close to 1 (i.e., $\lambda_l \sim \lambda_1 = 1$), then Θ_l and the overall error can be large.

Otherwise, the value of Θ_l depends mostly on the magnitude of the term $\delta\mathbf{M}\vec{x}'$ in the numerator, which can be summarized as a vector, $\vec{\epsilon}$, where all entries except for those corresponding to nodes that have incoming edges from the dummy node are 0. For those nodes that have incoming edges from the dummy node, $\vec{\epsilon}$ records the steady-state probability of the dummy node, multiplied with the error in the corresponding edge transition probabilities. Consequently, $\vec{\epsilon}$ consists mostly of 0s and values close to 0; when this vector is multiplied by $\vec{x}_l^\top$ (which is transpose of a vector normalized to 1) to obtain the contribution of the eigenvector $\vec{x}_l$ to the overall error, $\delta\vec{x}$, this will result in a small Θ_l – and hence a small error contribution.

2.5.2. Accuracy Loss due to the Use of Low Rank Approximation

As described earlier, the second source of loss of accuracy is due to the use of low rank approximation when decomposing the compensation matrix $\mathbf{M}_0$, for example (without loss of generality) using the graph partitioning approach presented by Tong et al (2006b). In their paper authors argue that, *under certain strict conditions*, given an $N \times N$ transition matrix $\mathbf{W}$, if a low rank approximation, $\hat{\vec{\pi}}$, of the corresponding personalized PageRank scores, $\vec{\pi}$, is obtained by partitioning the corresponding graph into t partitions, then the corresponding error, $\|\vec{\pi} - \hat{\vec{\pi}}\|_2$, can be shown to be inversely proportional to $1 - \lambda_i$ (for $i > 1$) where λ_i is the ith largest eigenvalue of $\mathbf{W}$. While this observation is well aligned with our error analysis described in the previous section, the conditions described in (Tong et al, 2006b) do not perfectly reflect our case. Therefore, in this subsection, we will reanalyze the error due to the low rank approximation of the compensation matrix $\mathbf{M}_0$.

Recall from previous section that the personalized PageRank scores, $\vec{\pi}$, given transition matrix $\mathbf{M}$ can be written as

$$\vec{\pi} = (1-\beta)\mathbf{M}\vec{\pi} + \beta\vec{s},$$

where β is the re-seeding probability and $\vec{s}$ is the re-seeding vector. In approach, we replace the original transition matrix $\mathbf{M}$ with

$$\mathbf{M}_{apx} = \mathbf{M}_{bd} + \mathbf{M}_0,$$

where $\mathbf{M}_{apx}$ is the transition matrix obtained by collapsing nodes in V^- into a single dummy node, $\mathbf{M}_{bd}$ is a block diagonal matrix and $\mathbf{M}_0$ is a compensation matrix. The low rank approximation is then applied on $\mathbf{M}_0$. Let us denote the perturbation caused by the low rank approximation on $\mathbf{M}_0$ as $\delta\mathbf{M}_0$; i.e., instead of $\mathbf{M}_{apx}$, we need to use

$\mathbf{M}'_{apx}$, where

$$\mathbf{M}'_{apx} = \mathbf{M}_{bd} + \mathbf{M}_0 + \delta\mathbf{M}_0 = \mathbf{M}_{apx} + \delta\mathbf{M}_0.$$

Consequently, we can write the equation for the personalized PageRank scores as

$$\vec{\pi}' = (1-\beta)\left(\mathbf{M}_{apx} + \delta\mathbf{M}_0\right)\vec{\pi} + \beta\vec{s}.$$

Note that, in the extreme case, the perturbation $\delta\mathbf{M}_0$ may be such that it isolates the seed nodes (by removing all the edges from seeds to the others), effectively nullifying the transition matrix, resulting in $\vec{\pi}' = \beta\vec{s}$. In general, the more the perturbations on the transition matrix due to the low rank approximation of $\mathbf{M}_0$ are correlated with the seeds, the higher will be the error. Therefore, in the rest of the discussion, we will focus on the error independent of the seeds; i.e., we will aim to quantify $\delta\vec{x} = \vec{x} - \vec{x}'$, where

$$\vec{x} = \mathbf{M}_{apx}\vec{x} \quad \text{and} \quad \vec{x}' = (\mathbf{M}_{apx} + \delta\mathbf{M}_0)\vec{x}'.$$

Note that this formulation of the error due to the low rank approximation, as a perturbation on the transition matrix, is analogous to the formulation of the error due to the clustering of the nodes in V^- into a single node discussed in the previous subsection. In both cases, the transition matrix is perturbed with a small perturbation matrix – the difference between the two cases is that in the case discussed in the previous section, the perturbations are structurally localized in the graph and are guaranteed to result in a random walk matrix, whereas in the case of perturbations due to low rank approximation, perturbations may be distributed across the entire matrix and $\mathbf{M}_{apx} + \delta\mathbf{M}_0$ may not be a random walk matrix. Given these, (following similar steps) we can derive the following expression for the error, $\delta\vec{x}$:

$$\delta\vec{x} \sim \sum_{l=2}^{R} \frac{\vec{x}_l^{\top}(\delta\mathbf{M} - \delta\lambda_1)\vec{x}'}{1-\lambda_l}\vec{x}_l,$$

where R is the number of eigenvectors of $\mathbf{M}_{apx}$ and λ_l is the lth largest eigenvalue and $\vec{x}_l$ is the corresponding eigenvector.

Note that, unlike the error term in the previous subsection, here the term $\delta\lambda_1$ may not disappear; this is because, while $\mathbf{M}_{apx}$ has its largest eigenvalue equal to 1, this may not be the case for $\mathbf{M}_{apx} + \delta\mathbf{M}_0$, which may not be a random walk matrix.

From this formulation of the error vector, $\delta\vec{x}$, we can deduce the following: First of all, as before (and as also observed by Tong et al (2006b), though under different conditions), the magnitude of the error depends on the separation of the largest eigenvalue (i.e., 1) and the other eigenvalues of the transition matrix.

Secondly, the magnitude of the error depends on the magnitude of the term $\delta\mathbf{M}_0\vec{x}'$ in the numerator. Since the entries in $\vec{x}'$ are likely to be larger for the nodes with high PageRank scores in the graph defined by $\mathbf{M}_{apx}$, this implies that the magnitude of the error depends on how correlated the perturbations on the compensation matrix (due to its low rank approximation) are with the nodes with high PageRank scores. In general, however, if the compensation matrix is low rank, we expect a small $\delta\mathbf{M}_0$ (which consists mostly of 0s and values close to 0) which would lead to a small overall error.

3. Complexity and Re-use

We can divide the work underlying the LR-PPR algorithm into five sub-tasks, each processed using only local nodes and edges:

Sub-task 1. The *preparatory* step in which the localities of the seeds are identified. The computational cost of this depends on the definition of locality. But, in general, the cost of this is linear in the size of the network G^+; i.e., $O(\|G^+\|)$, where $\|G^+\| \ll \|G\|$. Note that the work in this sub-task is entirely *re-usable*.

Next, the combined local transition matrix, $\mathbf{M}_{bd}$, and the compensation matrix, $\mathbf{M}_0$, are computed:

Sub-task 2a. Assuming a sparse matrix representation, computation and storage of the combined local transition matrix, $\mathbf{M}_{bd}$, takes $O(\sum_{1\leq l\leq K} \|G_l\|)$ time and space. Note that (while the matrix $\mathbf{M}_{bd}$ is not re-usable, unless the same set of seeds are provided) the constituting matrices $\mathbf{M}_1$ through $\mathbf{M}_K$ are *re-usable*.

Sub-task 2b. With a sparse representation, computation and storage of the compensation matrix takes $O(K \times max\_in\_degree \times \|V\| + (\|E\| - \sum_{1\leq l\leq K} \|E_l\|))$ time and space:

1. *Row/column indexing:* This takes $O(\sum_{1\leq l\leq K} \|V_l\|)$ time.
2. *Identification of common nodes (i.e., equivalence classes):* To locate the common nodes and to create the equivalence classes, we need to go over each node once and see if the node occurs in which of the remaining $K-1$ localities. Thus, assuming a hash-based implementation, this step takes $O(\sum_{1\leq l\leq K} \|V_l\|)$ to identify the equivalence classes.
3. *Identification of outgoing boundary edges:* In order to identify the outgoing boundary edges, we go over the nodes in V_1 through V_K and check if their outgoing edges are to a node within the same locality or not. If not, we check whether it is to a node within V^+ or not; if it is to a node in V^+, then the edge is labeled as an outgoing boundary edge among localities, otherwise, it is labeled as an outgoing boundary edge to G^-. Assuming that the nodes are labeled with their equivalence classes in the previous step, the cost of this operation is $O(\sum_{1\leq l\leq K} \sum_{v\in V_l} out(v))$.
   Note that, while the sub-task as a whole is not re-usable when the seed set changes, the part of the work involving identification of the outgoing edges from an individual locality is *re-usable*.
4. *Identification of incoming boundary edges from G^-:* In order to identify the incoming boundary edges from G^-, we go over the nodes in V_1 through V_K and check if their incoming edges are from a node marked with an equivalence class label. If not, the edge is from a node in G^-. The cost of this operation is $O(\sum_{1\leq l\leq K} \sum_{v\in V_l} in(v))$.
   Note that, while the sub-task as a whole is not re-usable when the seed set changes, the part of the work involving identification of the incoming edges into a single individual locality from nodes outside of the locality is *re-usable*.
5. *Compensation for the common nodes:* Once the $\|V^+\|$ equivalence classes are identified, the edges in the localities' incoming edges need to be rerouted (at most K times), leading to $O(K \times \sum_{1\leq l\leq K} \|E_l\|)$ time cost in the worst case.
6. *Compensation for the outgoing boundary edges:* This step involves considering once each outgoing boundary edge. Since all necessary information can be collected during the earlier identification pass (Subtask 2b. 3), the worst case time complexity of this operation is the same as that of the corresponding identification step.

7. *Compensation for the incoming boundary edges:* This step involves considering once each incoming boundary edge identified earlier. For each vertex, v, with one or more incoming edges, we create an edge whose weight captures the out-degrees of all corresponding external source nodes. Assuming that all nodes in the graph have been annotated with their out-degrees during a pre-processing step, the worst-case time complexity is the same as that of the corresponding identification step (Subtask 2b. 4).

8. *Compensation for the edges in G^-:* In the first look, it appears that this step cannot be executed without considering all nodes in V^-. However, this is not true: First of all, assuming that we know $\|V\|$, we can compute $\|V^-\|$ using $\|V\|$ and $\|V^+\|$. Secondly, the term $\sum_{v \in V^-} out(G^-, v)/out(G, v)$ can be rewritten as

$$\sum_{v \in V} \frac{out(G, v)}{out(G, v)} - \sum_{v \in V^+} \frac{out(G^+, v)}{out(G, v)} - \sum_{\substack{\langle v \to v_j \rangle \in \\ (inbound(G^+) \cup \\ outbound(G^+))}} \frac{1}{out(G, v)},$$

where $inbound(G^+)$ and $outbound(G^+)$ are the incoming and outgoing edges to G^+, both of which have been computed in earlier steps. Also, the first term is simply $\|V\|$. Thus, this step can be computed using only local information, in worst-case time complexity the same as the identification steps (Subtask 2b. 3 and Subtask 2b. 4).

Sub-task 3. Next, the $\mathbf{Q}_{bd}^{-1}$ matrix is obtained. The execution cost of this step is $O(\sum_{1 \leq l \leq K} matrix\_inversion\_cost(\mathbf{M}_l))$. There exists a $O(n^{2.373})$ algorithm for matrix inversion (Williams, 2011), where $n \times n$ is the dimensions of the input matrix. Thus, we can rewrite the execution cost as $O(\sum_{1 \leq l \leq K} \|V_l\|^{2.373})$. Assuming a sparse matrix representation, we need $O(\sum_{1 \leq l \leq K} \|V_l\|^2)$ space to store the resulting matrix $\mathbf{Q}_{bd}^{-1}$. Note that the work in this sub-task is, again, entirely *re-usable*.

Sub-task 4. Next, the compensation matrix, $\mathbf{M}_0$ is decomposed. While exact matrix decomposition is expensive, we use highly efficient approximate low-rank (r) decomposition (Tong et al, 2006b), which leverages sparsity of $\mathbf{M}_0$, where r is the selected rank.

Sub-task 5. The matrix, $\mathbf{\Lambda}$, is obtained. The matrix multiplications and inversions in this step take $O(r^{2.373} + r \times \|V^+\|^2 + r^2 \times \|V^+\|)$ time, where r is the selected rank.

Sub-task 6. Finally, $\vec{\phi}_{apx}$ of PPR scores is computed through matrix multiplications in $O(r \times \|V^+\|^2 + r^2 \times \|V^+\|)$ time.

Summary. This cost analysis points to the following advantages of the LR-PPR: First of all, computation is done using only local nodes and edges. Secondly, most of the results of the expensive sub-tasks 1, 2, and 3 can be cached and re-used. Moreover, costly matrix inversions are limited to the smaller matrices representing localities and small matrices of size $r \times r$.

It is important to note that various subtasks have complexity proportional to $\|V^+\|^2$, where $\|V^+\| = \sum_{1 \leq l \leq K} \|V_l\|$. While in theory the locality V_l can be arbitrarily large, in practice we select localities with a bounded number of nodes; i.e., $\forall_{1 \leq l \leq K}, \|V_l\| \leq L$ for some $L \ll \|V\|$.

4. Optimizations

The LR-PPR scheme involves: (a) initialization (where localities are identified and the local transition and compensation matrices are computed); (b) local transition matrix inversion, and (c) compensation matrix decomposition, $r \times r$ matrix inversion, and PPR computation. As mentioned above, tasks for (a) and (b) are cacheable and re-usable, whereas decomposition needs to be executed in query time. In this section, we discuss various optimization and parallelization opportunities.

4.1. Locality Selection

In the initialization phase of the algorithm, the first task is to identify localities for the given seed nodes (if they are not already identified and cached). A *locality graph* consists of a set of graph nodes that are *nearby* or otherwise *related* to a seed node. Note that localities can be *distance*-constrained or *size*-constrained. Common definitions include h-hop neighborhoods (Boldi et al, 2011; Cohen et al, 2003; Wei, 2010; Xiao et al, 2009; Zhou et al, 2009), reachability neighborhoods (Cohen et al, 2003), cluster/partition neighborhoods (Feige et al, 2005; Karypis and Kumar, 1998; Newman, 2006), or hitting distance neighborhoods (Chen et al, 2008; Mei et al, 2008). One straight-forward way to identify the locality of a seed node n is to perform breadth-first search around n to locate the closest L nodes in linear time to the size of the locality. Alternatively, one can use neighborhood indexing algorithms, such as `INI` (Kim et al, 2012), to identify the neighborhood of a given node in a way that captures topological characteristics (e.g., density of the edges) of the underlying graph.

4.2. Caching

As described above LR-PPR algorithm supports caching and re-use of some of the intermediary work. Sub-tasks 1 and 2 result in local transition matrices, each of which can be cached in $O(\|E_l\|)$ space (where E_l is the number edges in the locality) assuming a sparse representation. Sub-task 3, on the other hand, involves a matrix inversion, which results in a dense matrix; as a result, caching the inverted matrix takes $O(\|V_l\|^2)$ space (where V_l is the number of vertices in the locality). If the locality is size-constrained, this leads to constant space usage of $O(L^2)$, where L is the maximum number of nodes in the locality. If the inverted matrix of a locality is cached, then the local transition matrix does not need to be maintained further. Once the cache-space is full, we need to either push the cached inverted matrices into the secondary storage or drop some existing cached results from the memory. For cache replacement, any frequency-based or predictive cache-replacement policy can be used.

4.3. Parallelization Opportunities

Sub-task 1, which involves identifying localities of the seeds, is highly parallelizable: each seed can be assigned to a different processing unit; and the locality search can be parallelized through graph partitioning. If being leveraged, the `INI` algorithm (which relies on hash signatures) is highly parallelizable through signature partitioning (Kim et al, 2012). Sub-task 2, which involves construction of the local transition matrices and the compensation matrix is also parallelizable. Different localities and edges can be

Table 1. Data sets

Data Set	Overall Graph Characteristics		Locality Graph Characteristics	
	# nodes	# edges	# nodes per neighborhood	# edges per neighborhood
Epinions	∼76K	∼500K	from ∼200 to ∼2000	from ∼10K to ∼75K
SlashDot	∼82K	∼870K	from ∼700 to ∼5000	from ∼10K to ∼75K
WikiTalk	∼2.4M	∼5M	from ∼700 to ∼6000	from ∼10K to ∼75K
LiveJournal	∼4.8M	∼69M	from ∼900 to ∼6000	from ∼10K to ∼75K

Data Set	Seeds	
	# seeds	seed distances (hops)
Epinions	2-3	3-4
SlashDot	2-3	3-4
WikiTalk	2-3	3-4
LiveJournal	2-3	3-4

mapped to different servers for parallel processing. Sub-task 3, which involves matrix inversion of the local transition matrices is also parallelizable: different local matrices can be assigned to different processors; moreover, each matrix inversion itself can be parallelized (Pease, 1967). Sub-task 4 involves decomposition of the compensation matrix $\mathbf{M}_0$. Since $\mathbf{M}_0$ is sparse, this step can also be parallelized effectively (Gupta et al, 1997). Finally, Sub-tasks 5 and 6 involve matrix multiplications and inversions. As discussed above, matrix inversion operation can be parallelized. Similarly, there are well-known classical algorithms for parallelizing matrix multiplication (Gunnels et al, 1998).

5. Experimental Evaluation

In this section, we present results of experiments assessing the efficiency and effectiveness of the *Locality-Sensitive, Re-use Promoting Approximate Personalized PageRank* (LR-PPR) algorithm. Table 1 provides overviews of the four data sets (from `http://snap.stanford.edu/data/`) considered in the experiments. We considered graphs with different sizes and edge densities. We also varied numbers of seeds and the distances between the seeds (thereby varying the overlaps among seed localities). We also considered seed neighborhoods (or localities) of different sizes.

Unless otherwise specified, experiments were carried out using a 4-core Intel Core i5-2400, 3.10GHz, machine with 1024 KB L2 cache size, 6144 KB L3 cache size, 8GB memory, and 64-bit Windows 7 Enterprise. Codes were executed using Matlab 7.11.0(2010b). All experiments were run 10 times and averages are reported.

5.1. Alternative Approaches

In this section, we consider the following approaches to PPR computation:

- **Global PPR:** This is the default approach where the entire graph is used for PPR computation. We compute the PPR scores by solving the equation presented in Section 1.1.
- **FastRWR:** This is an approximation algorithm, referred to as NB_LIN in (Tong et al, 2006b). The algorithm reduces query execution times by partitioning the graph into subgraphs and preprocessing each partition. The pre-computed files are stored on disk and loaded to the memory during the query stage.
- **GMRES-PPR:** This is a recent alternative for computing PPR scores effi-

ciently (Maehara et al, 2014). We compare the PPR results obtained by our proposed algorithms' to those provided by GMRES-PPR[5], both in execution time and accuracy.

- **L-PPR:** This is our locality sensitive algorithm, where instead of using the whole graph, we use the *localized graph* created by combining the locality nodes and edges as described in Section 2.2. Once the localized transition matrix is created, the PPR scores are computed by solving the equation presented in Section 1.1.
- **LR-PPR:** This is the locality sensitive and re-use promoting algorithm described in detail in Section 2.4.

In the experiments, we set the restart probability, β, to 0.15 for all approaches.

5.2. Evaluation Measures

We consider three key evaluation measures:

- **Efficiency:** This is the amount of time taken to load the relevant (cached) data from the disk plus the time needed to carry out the operations to obtain the PPR scores.
- **Accuracy:** For different algorithm pairs, we report the Spearman's rank correlation

$$\frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}},$$

  which measures the agreement between two rankings (nodes with the same score are assigned the average of their positions in the ranking). Here, x and y are rankings by two algorithms and $\bar{x}$ and $\bar{y}$ are average ranks. To compute the rank coefficient, a portion of the highest ranked nodes in the merged graph according to x are considered. As default, we considered 10% highest ranked nodes; but we also varied the target percentage (5%, 10%, 25%, 50%, 75%) to observe how the accuracy varies with result size.
- **Memory:** We also report the amount of data read from the cache.

5.3. Results and Discussions

5.3.1. Proposed Algorithms (L-PPR and LR-PPR) versus FastRWR and GMRES-PPR

In our experiments, we compared the proposed algorithms against FastRWR and GMRES-PPR.

Settings. GMRES-PPR takes a bag size (d) as input parameter. Authors show that the algorithm runs faster when the value of d is smaller. In our experiments, we set $d = 10$ – a relatively small value that leads to fast executions for GMRES-PPR as shown in (Maehara et al, 2014). Unfortunately, even with this small value of d, GMRES-PPR did not successfully complete its preprocessing phase. In particular, the LU step of the process failed for the large data set, even when we repeated the experiment with an alternative, substantially better hardware involving an 8-core Intel Core i7-4770, 3.40 GHz machine with 32.0 GB memory and 1024 L2Cache and 8192 Cache size.

Similarly, FastRWR takes as input the number, t, of partitions, which impacts its

[5] We obtained the source code for the preprocessing phase for this algorithm from the authors.

Table 2. **FastRWR settings and impact on its performance:** In our experiments, we compared the proposed algorithms against FastRWR (Tong et al, 2006b). To ensure that our comparison is fair to FastRWR (Tong et al, 2006b), we selected the number of FastRWR partitions in a way that minimizes its execution time and memory and maximizes its quality. This table shows the FastRWR performance for different data sets and configurations; the bold entries correspond to the high accuracy low time and memory configuration selected for the experiments in this section.

Data Set	# part.	Time (sec.)		Top-10	Memory
		Disk I/O	In-Memory	Sp. Correl.	(MB)
Epinions	3	18.02	0.58	0.96	1547
∼76K nodes	**40**	**0.22**	**0.04**	**0.97**	**178**
∼500K edges	400	0.15	0.03	0.95	140
	1000	0.16	0.02	0.95	138
SlashDot	3	Out of memory in Q_1^{-1} calculation			
∼82K nodes	10	0.79	0.23	0.96	616
∼870K edges	**40**	**0.40**	**0.08**	**0.96**	**302**
	400	0.27	0.05	0.92	244
	1000	0.28	0.04	0.95	250
WikiTalk	3	Out of memory in Q_1^{-1} calculation			
∼2.4M nodes	40	Out of memory in Q_1^{-1} calculation			
∼5M edges	200	Out of memory in Q_1^{-1} calculation			
	400	24.03	17.60	0.86	1454
	1000	**16.75**	**15.15**	**0.87**	**1429**
LiveJournal	1000	Out of memory in $\hat{\Lambda}$ calculation			
∼4.8M nodes	3000	Out of memory in $\hat{\Lambda}$ calculation			
∼69M edges	5000	Out of memory in $\hat{\Lambda}$ calculation			

execution time, query time memory usage, as well as approximation quality. As shown in Table 2, in our experiments, to be fair against FastRWR, we selected the number of its partitions in a way that minimizes its execution time and memory and maximizes its quality. Note that, especially for large data sets, FastRWR requires large number of partitions to ensure that the intermediary metadata (which requires dense matrix representation) fits into the available memory (8GB) and this negatively impacts accuracy. For the LiveJournal data set, even with large number of partitions, the pre-computation stage of FastRWR could not be completed due to memory requirements. To see whether this phase could complete with a machine with better hardware specifications, we also repeated the experiment with an alternative 8-core Intel Core i7-4770, 3.40 GHz machine with 32.0 GB memory and 1024 L2Cache and 8192 Cache size. Unfortunately, FastRWR ran out of memory even with this substantially better hardware.

Execution Time: Table 3 presents execution time results for FastRWR, GMRES-PPR, L-PPR, LR-PPR, as well as Global PPR on two different locality graph sizes ($10K$ and $75K$ edges, respectively).

First of all, we see that all four algorithms are much faster than Global PPR. As Table 3 shows, when the seed locality graph size is small, L-PPR and LR-PPR significantly outperform FastRWR and GMRES-PPR.

We also see that, the major contributor on the execution times of L-PPR and LR-PPR is not the size of the whole graph but the size of the merged network of localities. Since locality graphs are generally small relative to the size of the whole graph, L-PPR and LR-PPR can be calculated very efficiently.

In *small data sets* (Epinions and Slashdot) FastRWR and GMRES-PPR work slightly faster than L-PPR and LR-PPR when we consider the large locality size (75K edges per locality) In *large data sets* (WikiTalk), however, both L-PPR and LR-PPR

Table 3. Execution time results for different algorithms and configurations

Data set	Seeds		Merged Network		Execution Time (sec.)				
	# seeds	# hops	Avg # nodes	Avg # edges	Global PPR	Fast RWR	GMRES-PPR	L-PPR	LR-PPR
Seed Localities with ∼10K Edges									
Epinions ∼76K nodes ∼500K edges	2	3	∼0.7K	∼17K	26.44	0.20	0.12	0.05	0.03
	2	4	∼0.6K	∼15K	28.06	0.21	0.12	0.05	0.04
	3	3	∼0.7K	∼19K	30.40	0.22	0.12	0.07	0.04
	3	4	∼0.8K	∼20K	30.36	0.22	0.12	0.17	0.05
SlashDot ∼82K nodes ∼870K edges	2	3	∼1.3K	∼15K	21.56	0.34	0.20	0.08	0.07
	2	4	∼1.9K	∼17K	21.96	0.34	0.18	0.08	0.07
	3	3	∼1.8K	∼19K	22.25	0.35	0.18	0.10	0.09
	3	4	∼2.5K	∼25K	22.54	0.35	0.19	0.15	0.10
WikiTalk ∼2.4M nodes ∼5M edges	2	3	∼4.1K	∼19K	677.32	17.18	0.39	0.23	0.21
	2	4	∼4.8K	∼20K	741.08	16.51	0.40	0.29	0.26
	3	3	∼4.4K	∼24K	709;35	16.71	0.42	0.34	0.31
	3	4	∼5.2K	∼29K	763.10	16.61	0.41	0.37	0.21
LiveJournal ∼4.8M nodes ∼69M edges	2	3	∼2.0K	∼19K	-	-	-	0.16	0.17
	2	4	∼0.9K	∼20K	-	-	-	0.24	0.22
	3	3	∼3.0K	∼30K	-	-	-	0.21	0.19
	3	4	∼1.0K	∼30K	-	-	-	0.26	0.18
Seed Localities with ∼75K Edges									
Epinions ∼76K nodes ∼500K edges	2	3	∼2.2K	∼90K	26.44	0.21	0.12	0.37	0.14
	2	4	∼3.0K	∼99K	27.58	0.22	0.12	0.51	0.20
	3	3	∼2.7K	∼108K	27.30	0.21	0.12	0.58	0.26
	3	4	∼3.5K	∼120K	27.90	0.22	0.12	0.76	0.36
SlashDot ∼82K nodes ∼870K edges	2	3	∼5.9K	∼117K	21.79	0.35	0.20	0.70	0.53
	2	4	∼5.7K	∼125K	21.85	0.35	0.18	0.78	0.42
	3	3	∼7.1K	∼141K	21.74	0.36	0.18	1.12	0.95
	3	4	∼7.2K	∼159K	22.93	0.38	0.19	1.39	0.83
WikiTalk ∼2.4M nodes ∼5M edges	2	3	∼5.7K	∼102K	681.08	16.28	0.39	0.75	0.37
	2	4	∼5.8K	∼100K	693.44	16.22	0.40	0.73	0.37
	3	3	∼6.3K	∼101K	701.34	16.32	0.42	0.75	0.37
	3	4	∼6.7K	∼103K	706.26	16.34	0.41	0.78	0.36
LiveJournal ∼4.8M nodes ∼69M edges	2	3	∼7.9K	∼144K	-	-	-	1.66	0.83
	2	4	∼2.9K	∼149K	-	-	-	1.06	0.32
	3	3	∼9.8K	∼207K	-	-	-	3.05	1.01
	3	4	∼4.8K	∼213K	-	-	-	2.63	0.57

significantly outperform FastRWR and LR-PPR takes less time than GMRES-PPR in terms of query processing efficiency. An important observation is that, as described earlier, the preprocessing phases of FastRWR and GMRES-PPR did not complete for the *very large* LiveJournal data set, even on significantly boosted hardware setups. Our proposed algorithms, however, successfully completed the preprocessing phase in the original set up and provided very fast query execution times as reported in the table.

Memory Usage: As we see in Table 4, GMRES-PPR, L-PPR, and LR-PPR, all provide significant gains relative to FastRWR in terms of memory usage.

When the localities consist of ~ 10K edges, LR-PPR and GMRES-PPR have comparable memory requirements, while L-PPR outperforms all in terms of the run-time memory needed during the on-line phase. When the localities are larger (75K edges), on the other hand, L-PPR and GMRES-PPR have comparable memory requirements, while LR-PPR needs more memory. As we see below, however, the lower memory requirements of GMRES-PPR relative to LR-PPR come with a significant drawback in terms of its accuracy relative to proposed algorithms.

Accuracy: Table 5 compares FastRWR, GMRES-PPR, L-PPR, and LR-PPR in terms of accuracy. As we see in this table, the proposed locality sensitive techniques, L-PPR and LR-PPR, constantly outperform FastRWR and GMRES-PPR and the accuracy gap is especially large in *large data sets*, such as WikiTalk. This is because, for FastRWR,

Table 4. Memory requirements for different algorithms and configurations

Data set	Seeds		Merged Network		Memory Usage (MB)			
	# seeds	# hops	Avg # nodes	Avg # edges	Fast RWR	GMRES-PPR	L-PPR	LR-PPR
Seed Localities with ∼10K Edges								
Epinions ∼76K nodes ∼500K edges	2	3	∼2.2K	∼90K	178.3	8.55	0.63	4.40
	2	4	∼3.0K	∼99K			0.71	5.69
	3	3	∼2.7K	∼108K			0.96	7.30
	3	4	∼3.5K	∼120K			1.03	8.09
SlashDot ∼82K nodes ∼870K edges	2	3	∼5.9K	∼117K	302.1	12.16	0.64	4.40
	2	4	∼5.7K	∼125K			1.43	16.66
	3	3	∼7.1K	∼141K			2.08	27.92
	3	4	∼7.2K	∼159K			2.17	23.36
WikiTalk ∼2.4M nodes ∼5M edges	2	3	∼5.7K	∼102K	1429.0	20.97	5.66	26.74
	2	4	∼5.8K	∼100K			5.51	31.44
	3	3	∼6.3K	∼101K			8.82	40.46
	3	4	∼6.7K	∼103K			8.49	76.08
LiveJournal ∼4.8M nodes ∼69M edges	2	3	∼2.0K	∼19K	-	-	1.70	23.55
	2	4	∼0.9K	∼20K			3.19	17.96
	3	3	∼3.0K	∼30K			3.25	37.97
	3	4	∼6.7K	∼103K			3.64	22.71
Seed Localities with ∼75K Edges								
Epinions ∼76K nodes ∼500K edges	2	3	∼2.2K	∼90K	178.3	8.55	2.9	36.3
	2	4	∼3.0K	∼99K			3.1	55.2
	3	3	∼2.7K	∼108K			4.6	57.6
	3	4	∼3.5K	∼120K			4.7	77.7
SlashDot ∼82K nodes ∼870K edges	2	3	∼5.9K	∼117K	302.1	12.16	5.0	228.1
	2	4	∼5.7K	∼125K			4.9	172.8
	3	3	∼7.1K	∼141K			7.6	325.9
	3	4	∼7.2K	∼159K			7.2	256.0
WikiTalk ∼2.4M nodes ∼5M edges	2	3	∼5.7K	∼102K	1429.0	20.97	15.5	114.5
	2	4	∼5.8K	∼100K			16.2	120.7
	3	3	∼6.3K	∼101K			24.0	211.6
	3	4	∼6.7K	∼103K			28.7	197.5
LiveJournal ∼4.8M nodes ∼69M edges	2	3	∼7.9K	∼144K	-	-	10.99	322.87
	2	4	∼2.9K	∼149K			8.24	68.10
	3	3	∼9.8K	∼207K			15.12	374.91
	3	4	∼4.8K	∼213K			13.48	138.25

it tries to approximate the whole graph, whereas the proposed algorithms focus on the relevant localities. As also discussed in Section 5.1, FastRWR requires large number of partitions to ensure that the intermediary metadata (which requires dense matrix representation) fits into memory and this negatively impacts accuracy. Our locality-sensitive algorithms, L-PPR and LR-PPR, avoid this and provide high accuracy with low memory consumption, especially in large graphs, like WikiTalk.

Figure 10 reconfirms that the accuracies of L-PPR and LR-PPR both stay high as we consider larger numbers of top ranked network nodes for accuracy assessment, whereas the accuracies of FastRWR and GMRES-PPR suffer significantly when we consider larger portions of the merged locality graph.

A Detailed Look at the Execution Times, Accuracies, and Memory Usage of the Alternative Approaches:

• *Epinions Data Set:* Figure 11 compares in further detail the execution times, accuracies, and amounts of data read by L-PPR, LR-PPR, FastRWR, and GMRES-PPR from the cache per query as a function of the size of the merged locality network for different seeds and target locality sizes of the Epinions data set. As the figure re-confirms, L-PPR and LR-PPR provide significantly higher accuracies than other algorithms. LR-PPR needs more space than L-PPR to fetch the cached localities for reuse, but it uses this memory effectively to significantly reduce the execution time. The figure also re-confirms the execution time results presented in Table 3: as the figure shows, the time

Table 5. Accuracy results for different algorithms and configurations: note that, since computation of the Global PPR (which is the ground truth for accuracy) is not feasible for the very large Live Journal data set, we only include accuracy computations for the other three data sets

Data set	Seeds		Merged Network		Top-10% Correl. (vs Global PPR)			
	# seeds	# hops	Avg # nodes	Avg # edges	Fast RWR	GMRES-PPR	L-PPR	LR-PPR
Seed Localities with ∼10K Edges								
Epinions ∼76K nodes ∼500K edges	2	3	∼0.7K	∼17K	0.954	0.826	0.990	0.988
	2	4	∼0.6K	∼15K	0.959	0.825	0.992	0.993
	3	3	∼0.7K	∼19K	0.958	0.823	0.991	0.986
	3	4	∼0.8K	∼20K	0.958	0.823	0.987	0.985
SlashDot ∼82K nodes ∼870K edges	2	3	∼1.3K	∼15K	0.921	0.810	0.984	0.958
	2	4	∼5.7K	∼125K	0.922	0.818	0.987	0.977
	3	3	∼1.8K	∼19K	0.921	0.813	0.973	0.973
	3	4	∼2.5K	∼25K	0.921	0.818	0.982	0.974
WikiTalk ∼2.4M nodes ∼5M edges	2	3	∼4.1K	∼19K	0.868	0.853	0.957	0.983
	2	4	∼4.8K	∼20K	0.871	0,854	0.994	0.984
	3	3	∼4.4K	∼24K	0.866	0.852	0.986	0.988
	3	4	∼5.2K	∼29K	0.855	0.852	0.973	0.964
Seed Localities with ∼75K Edges								
Epinions ∼76K nodes ∼500K edges	2	3	∼2.2K	∼90K	0.963	0.823	0.997	0.990
	2	4	∼3.0K	∼99K	0.960	0.824	0.998	0.990
	3	3	∼2.7K	∼108K	0.967	0.826	0.998	0.990
	3	4	∼3.5K	∼120K	0.967	0.825	0.997	0.991
SlashDot ∼82K nodes ∼870K edges	2	3	∼5.9K	∼117K	0.955	0.816	0.973	0.990
	2	4	∼5.7K	∼125K	0.943	0.816	0.965	0.983
	3	3	∼7.1K	∼141K	0.957	0.815	0.971	0.990
	3	4	∼7.2K	∼159K	0.958	0.815	0.976	0.986
WikiTalk ∼2.4M nodes ∼5M edges	2	3	∼5.7K	∼102K	0.868	0.851	0.958	0.944
	2	4	∼5.8K	∼100K	0.870	0.848	0.930	0.929
	3	3	∼6.3K	∼101K	0.877	0.852	0.937	0.927
	3	4	∼6.7K	∼103K	0.869	0.851	0.976	0.967

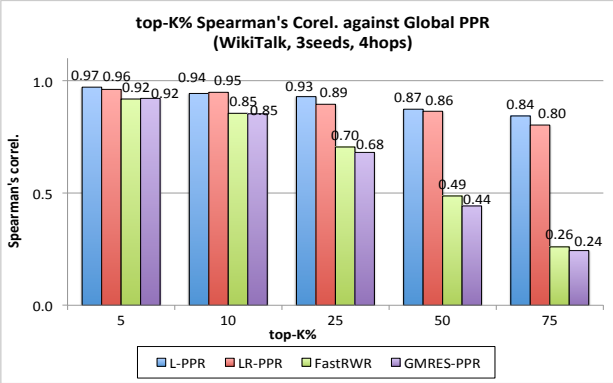
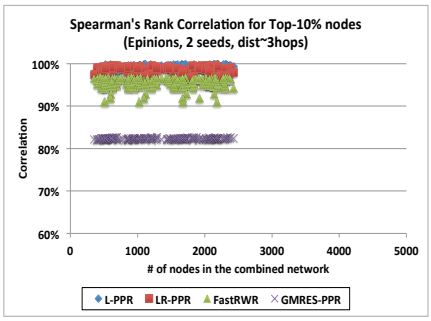
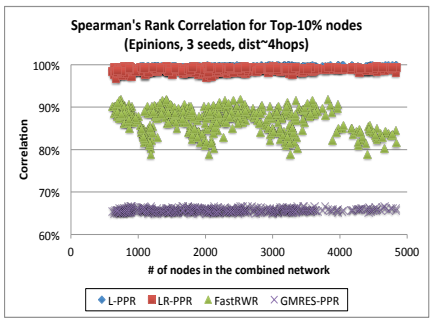


Fig. 10. Accuracies of L-PPR, LR-PPR, FastRWR, and GMRES-PPR against the Global PPR for different numbers of target nodes

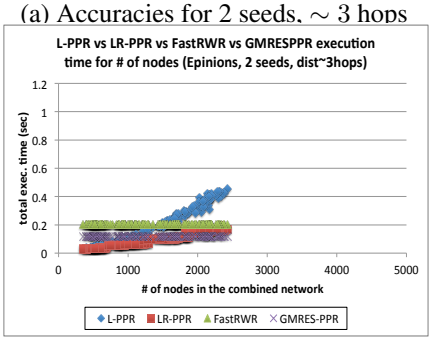
cost increases for all algorithms as the number of seeds increases; but, the cost of LR-PPR (which leverages re-use) increases much slower than the cost of L-PPR. In the case of the Epinions data set, FastRWR works slightly faster than LR-PPR for large numbers of seeds and larger neighborhoods; however, this comes with a significant loss in accuracy and also higher memory usage than L-PPR and LR-PPR. Note that, since FastRWR does not scale as well as L-PPR and LR-PPR with the overall graph size, this slight execution time advantage of FastRWR also disappears in the case of large graphs



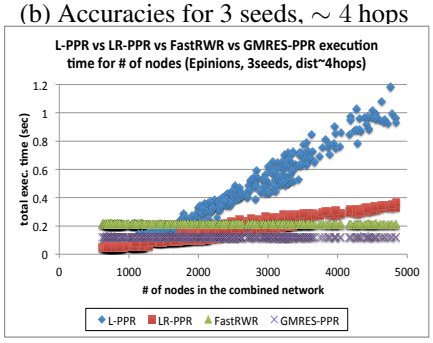
(a) Accuracies for 2 seeds, $\sim$ 3 hops



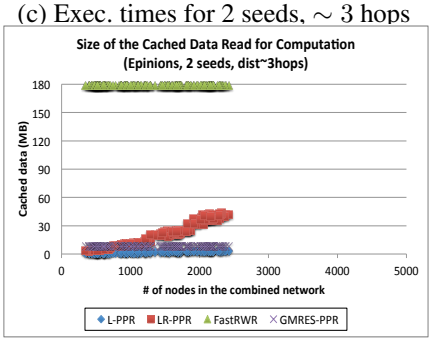
(b) Accuracies for 3 seeds, $\sim$ 4 hops



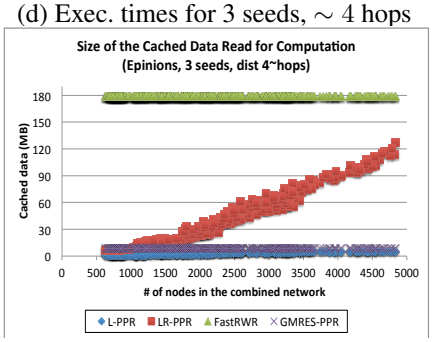
(c) Exec. times for 2 seeds, $\sim$ 3 hops



(d) Exec. times for 3 seeds, $\sim$ 4 hops



(e) Data for 2 seeds, $\sim$ 3 hops



(f) Data for 3 seeds, $\sim$ 4 hops

Fig. 11. Performances of L-PPR, LR-PPR, FastRWR, and GMRES-PPR as a function of the size of the combined localities network (Epinions data set) for different numbers of seeds, selected at varying hop distances from each other

like WikiTalk (as was seen in Table 3 and also summarized in Figure 12). L-PPR and LR-PPR may take more time than GMRES-PPR when the combined network of localities is large, however as we had also observed earlier, this comes with a significant loss in accuracy.

• *SlashDot Data Set:* The results for the SlashDot data set (which have similar graph structure as the Epinions data set; see Table 1) are similar to the Epinions results and, hence, presented in the Appendix.

• *WikiTalk Data Set:* The WikiTalk data set however has a different structure and, thus, we present the execution times, accuracies, and amounts of data read by L-PPR, LR-PPR, FastRWR, and GMRES-PPR for the WikiTalk data set in Figure 13. The most

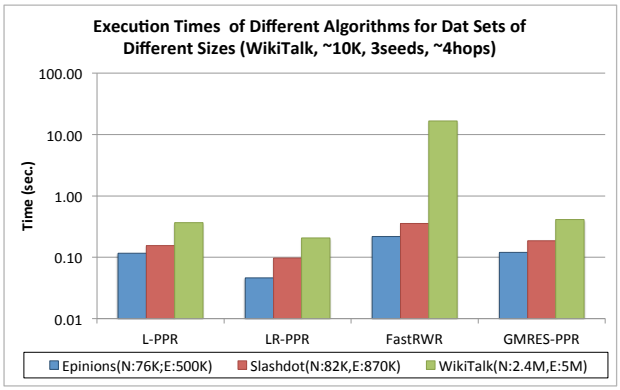


Fig. 12. Execution times of the algorithms L-PPR, LR-PPR, FastRWR, and GMRES-PPR for different data sets of varying sizes

important thing to recognize when comparing Figures 11 (for the Epinions data set) and 13 (for the WikiTalk data set) is that when the graph is larger (i.e., for the WikiTalk data set), the execution time gains of L-PPR and LR-PPR relative to other alglorithms are even more pronounced. Similarly, as the problem size gets larger (e.g., WikiTalk data, 3 seeds, $\sim$ 4 hops), the accuracy gains of L-PPR and LR-PPR relative to FastRWR and GMRES-PPR also become even more significant. This re-confirms that the proposed locality-sensitive (and re-use promoting) techniques provide not only better scalabilities, but also better accuracies than existing algorithms.

• *LiveJournal Data Set:* Since the overall pattern is similar to the others, we present the execution times and amounts of data read for the LiveJournal data set in the Appendix.

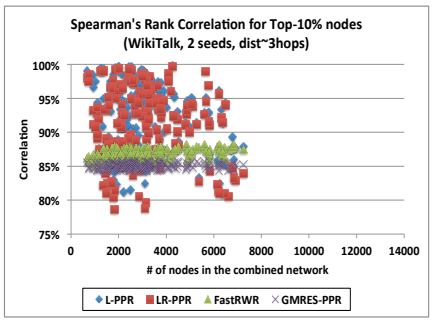
5.3.2. Detailed Analysis of L-PPR and LR-PPR

As we have seen in Figure 10 and Tables 3 through 5, locality-sensitive and re-use promoting LR-PPR constantly outperforms only locality-sensitive L-PPR ($\sim 1.5\times$ to $2\times$), while returning almost as accurate results.

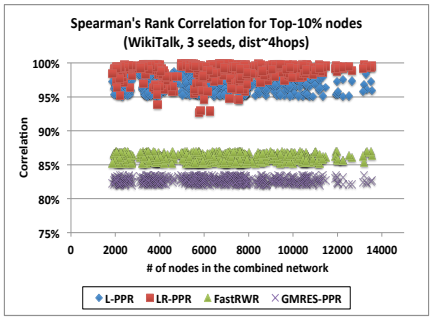
Distribution of the Execution Times of L-PPR **and** LR-PPR**.** Figure 14 further investigates how the execution times of L-PPR and LR-PPR are distributed among their sub-tasks. As predicted in Section 2.4, LR-PPR spends significant portions of its time in loading data from the cache, reindexing nodes, and creating compensation matrices. Creating the low-rank approximation of $\mathbf{M}_0$, computing the matrix $\mathbf{\Lambda}$, and solving for PPR scores take relatively little time.

Performances of LR-PPR as a Function of the Size of the Merged Locality Network. Figure 15 shows the execution times, accuracy, and amount of data read by LR-PPR from the cache per query as a function of the size of the merged locality network. As the figure shows, the execution time (Figure 15(a)) tracks the amount of data brought into the memory (Figure 15(b)), whereas the accuracy is relatively constant (Figure 15(c)).

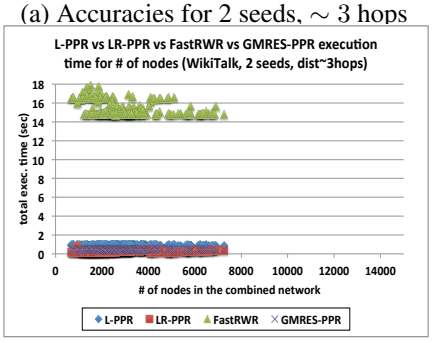
Impact of the Boundary Edges on the Performances of L-PPR and LR-PPR. Recall from Section 2.2.3, Figures 8 and 9, that the merged graph represents nodes outside of the seed localities using a single combined node, which is then connected to the nodes in the seed localities, with outgoing and incoming boundary edges. Figure 16 shows the



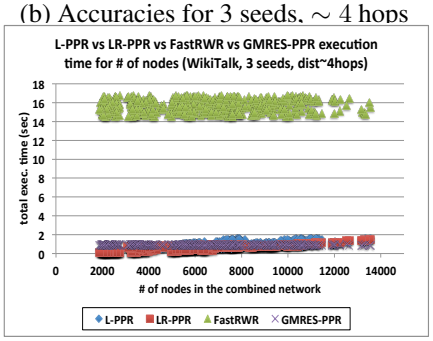
(a) Accuracies for 2 seeds, $\sim$ 3 hops



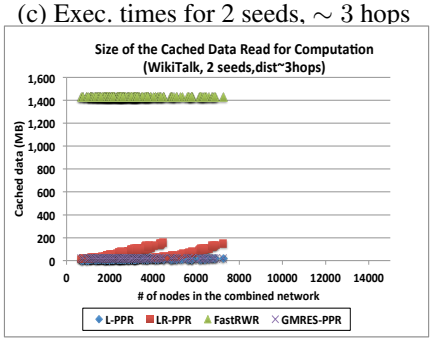
(b) Accuracies for 3 seeds, $\sim$ 4 hops



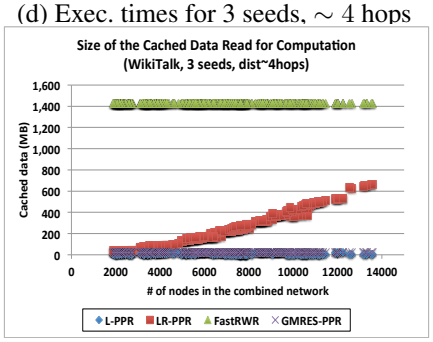
(c) Exec. times for 2 seeds, $\sim$ 3 hops



(d) Exec. times for 3 seeds, $\sim$ 4 hops



(e) Data for 2 seeds, $\sim$ 3 hops

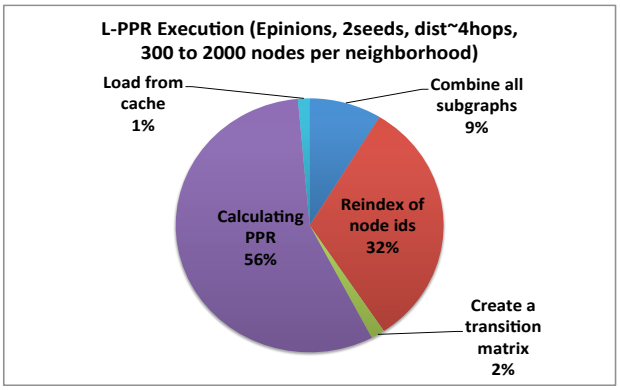


(f) Data for 3 seeds, $\sim$ 4 hops

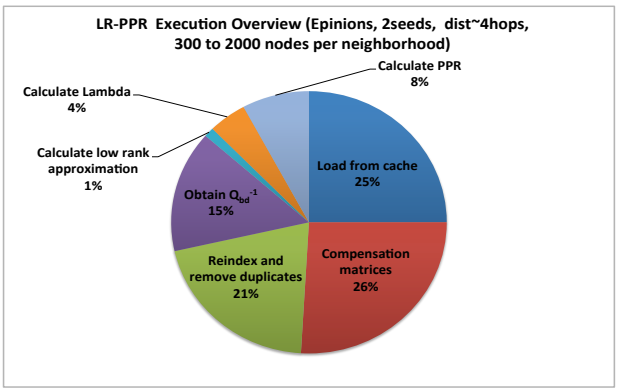
Fig. 13. Performances of L-PPR, LR-PPR, and FastRWR as a function of the size of the combined localities network (WikiTalk data set) for different numbers of seeds, selected at varying hop distances from each other

impact of the amount of edges at this boundary. As the figure shows, for a fixed merged locality graph size, the larger the number of boundary edges, the higher the execution times for both L-PPR and LR-PPR; moreover, the larger the merged graph, the faster the increase in the cost. However, the figure also shows that LR-PPR is much less affected from the boundary edges than the basic L-PPR.

Figure 17 confirms the impact of the boundary edges on a second data set. As we have seen in Tables 3 through Table 5, for the SlashDot data set, LR-PPR shows a slightly different behavior than for Epinions and WikiTalk data sets: while LR-PPR still outperforms basic L-PPR, the difference is smaller under some configurations. Figure 17(a) and (b) explain the reason in terms of the ratio of the boundary edges: in the



(a) L-PPR



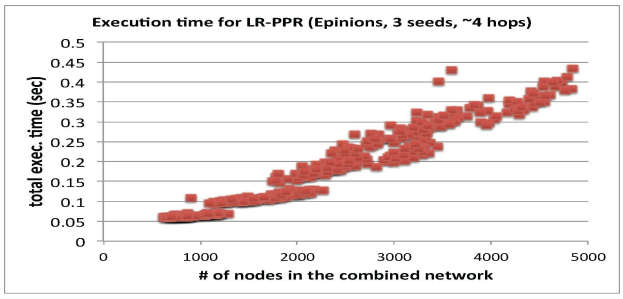
(b) LR-PPR

Fig. 14. Distribution of the execution times for L-PPR and LR-PPR for the Epinions data set

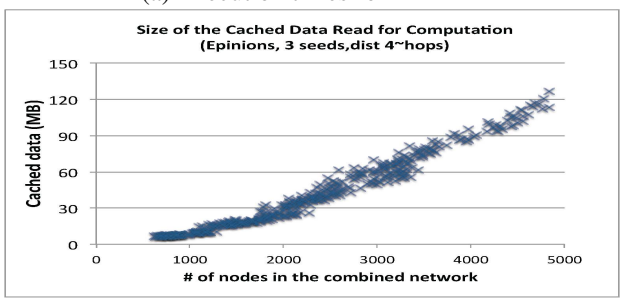
SlashDot data set, when the seeds are close (i.e., when localities overlap significantly), the boundary edges are relatively few and the impact of the boundary edges are similar for both LR-PPR and L-PPR; when the seeds are further away, on the other hand there are more boundary edges and LR-PPR's effectiveness in dealing efficiently with the boundary edges becomes more pronounced. Thus, since the accuracy is not affected and stays high for both LR-PPR and L-PPR, the ratio of the boundary edges in the merged graph can be used as an indicator for when to use LR-PPR and when to simply leverage basic locality-sensitive L-PPR.

Parallelization of the Off-line Process. As we see in Section 4.3, there are various opportunities for parallelization of the tasks involved in the off-line and on-line steps of L-PPR and LR-PPR.

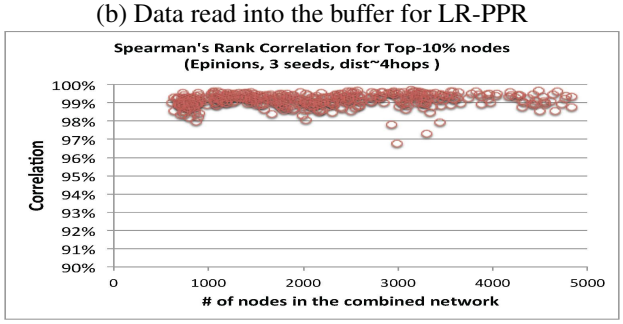
One of the costliest steps of the process is Sub-Task 1, which is the generation of the locality graphs and calculation of $\mathbf{Q}_h^{-1}$ for the seed nodes. Figure 18 shows the execution time of this sub-task with and without parallelization. In this figure, 'no-parallel' refers to the situation where all available cores of the machine are used during the calculation of this step, but without explicit parallelization of the steps. '1 core' to '4 cores' refer to scenarios where different numbers of cores are assigned to an explicitly



(a) Execution times for LR-PPR



(b) Data read into the buffer for LR-PPR



(c) Accuracy for LR-PPR

Fig. 15. Performance of LR-PPR as a function of the size of the combined localities network (Epinions data set, 3 seeds, ∼4 hops): execution time of the LR-PPR is proportional to the data read from the cache

parallelized implementation of this step. As we see in this figure, as we allocate more cores [6], explicitly parallelized implementations are able to pull the execution times down for all data sets and all locality sizes.

[6] For the Epinions, SlashDot, and WikiTalk data sets, we used a machine with 4 cores, 8GB memory, 1024 KB L2 cache, and the 6144 KB L3 cache. For the much larger LiveJournal data set, we used a machine with 8 cores, 32GB memory, 1024 KB L2 cache, and the 8192 KB L3 cache. With lesser cache space, the parallelization lost its effectiveness for the LiveJournal data set.

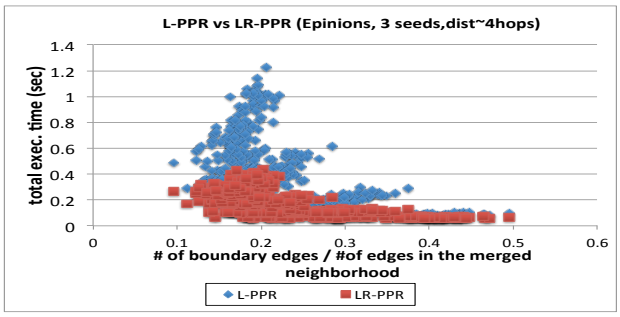
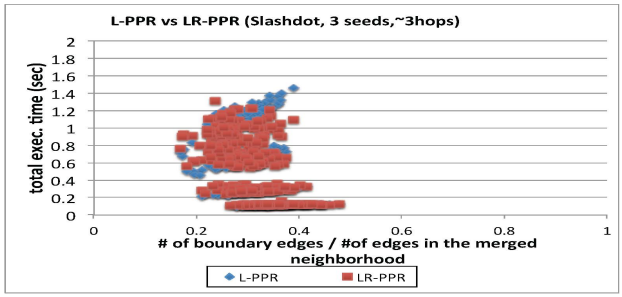
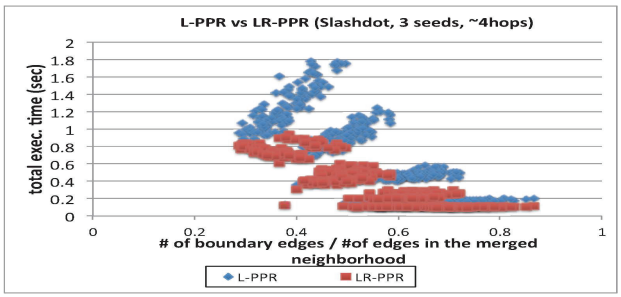


Fig. 16. The impact of the ratio of the boundary edges on the execution time for L-PPR and LR-PPR (Epinions, 3 seeds, with distance $\sim$ 4 hops): the larger the ratio of boundary edges, the higher the execution times for both L-PPR and LR-PPR; but, LR-PPR is less affected from the ratio of the boundary edges than the basic L-PPR



(a) effect of the boundary for 3 seeds, $\sim$3 hops

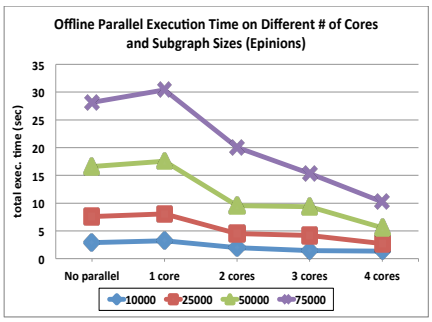


(b) effect of the boundary for 3 seeds, $\sim$4 hops

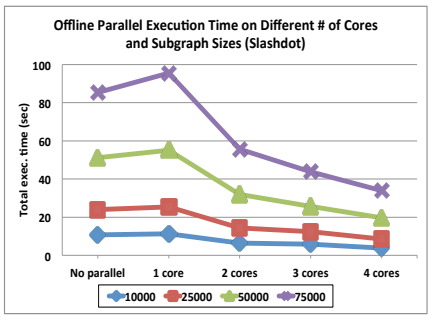
Fig. 17. Impact of the boundary edges for the SlashDot data set (3 seeds): note that in the SlashDot data set, when the seeds are close (a) the boundary edges are relatively fewer

6. Conclusions

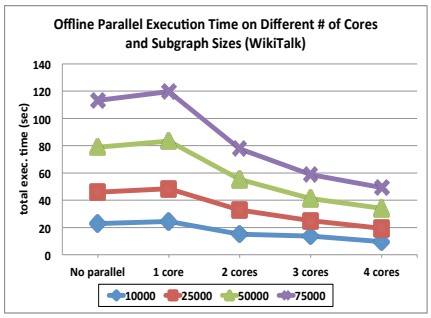
In this paper, we presented a *Locality-sensitive, Re-use promoting, approximate Personalized PageRank (LR-PPR)* algorithm for efficiently computing the PPR values relying on the localities of the seed nodes on the graph. Instead of performing a *monolithic* computation for the given seed node set using the entire graph, LR-PPR divides the work



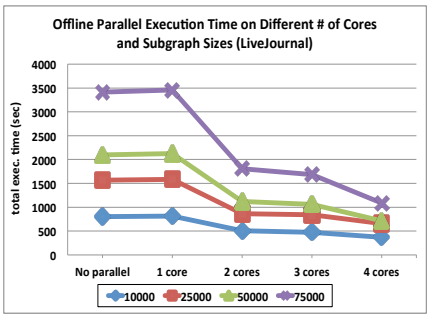
(a) Exec. times for Epinions data set



(b) Exec. times for Slashdot data set



(c) Exec. times for WikiTalk data set



(d) Exec. times for LiveJournal data set

Fig. 18. Execution time needed for generating locality graphs and calculating $\mathbf{Q}_h^{-1}$ on different number of cores (4 seeds, different curves corresponds to different locality sizes)

into localities of the seeds and caches the intermediary results obtained during the computation. These cached results can then be reused for future queries sharing seed nodes. Experiments showed that the proposed LR-PPR approach provides significant gains in execution time relative to existing approximate PPR computation techniques, where the PPR scores are computed from scratch using the whole network. LR-PPR also outperforms L-PPR, where the PPR scores are computed in a locality-sensitive manner, but without significant re-use, with negligible impacts on accuracy.

Acknowledgements. This work is supported by NSF Grants 1318788 "Data Management for Real-Time Data Driven Epidemic Spread Simulations" and 1339835 "E-SDMS: Energy Simulation Data Management System Software". A preliminary version of this work appeared as (Kim et al, 2013): *Jung Hyun Kim, K. Selçuk Candan, and Maria Luisa Sapino. LR-PPR: Locality-Sensitive, Re-use Promoting, Approximate Personalized PageRank Computation. ACM International Conference on Information and Knowledge Management (CIKM'13), October 2013.* We especially thank Leonardo Allisio and Ilario Dal Grande for their feedback and corrections on the manuscript and authors of (Maehara et al, 2014) for sharing with us their source code for the preprocessing stage of their algorithm.

References

Avrachenkov K, Litvak N, Nemirovsky D, Smirnova E, Sokol M (2011) Quick Detection of Top-k Personalized PageRank Lists. WAW'11, pp 50-61, 2011.

Bahmani B, Chakrabarti K, Xin D (2011) Fast personalized PageRank on MapReduce. In SIGMOD'11. 973-984. 2011.

Bahmani B, Chowdhury A, Goel A (2010) Fast incremental and personalized PageRank. PVLDB. 4, 3, 173-184, 2010.

Balmin A, Hristidis V, Papakonstantinou Y (2004) ObjectRank: Authority-based keyword search in databases. VLDB, 2004.

Boldi P, Rosa M, Vigna S (2011) HyperANF: Approximating the neighbourhood function of very large graphs on a budget. WWW'11, 2011.

Brin S, Page L (1998) The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30: 107-117, 1998.

Candan K.S and Li W.S (2000) Using random walks for mining web document associations. In *PAKDD*, pp. 294-305, 2000.

Candan K.S and Li W.S (2002) Reasoning for Web document associations and its applications in site map construction. Data Knowl. Eng. 43(2),2002.

Chakrabarti S (200) Dynamic personalized pagerank in entity-relation graphs. WWW '07, 2007.

Chen M, Liu J, and Tang X (2008) Clustering via random walk hitting time on directed graphs. AAAI'08, pp. 616-621, 2008.

Cohen E, Halperin E, Kaplan H, Zwick U (2003) Reachability and distance queries via 2-hop labels. SIAM Journal of Computing, vol. 32, no. 5, 2003.

Csalogany K, Fogaras D, Racz B, Sarlos T (2005) Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments Internet Math. 2,3, 333-358, 2005.

Feige U, Hajiaghayi M, Lee J.R (2005) Improved approximation algorithms for minimum-weight vertex separators. STOC, 2005.

Fouss F, Pirotte A, Renders J, Saerens M (2007) Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. TKDE, 2007.

Fujiwara Y, Nakatsuji M, Onizuka M, Kitsuregawa M (2012) Fast and exact top-k search for random walk with restart. PVLDB. 5, 5, 442-453. 2012.

Gunnels J, Lin C, Morrow G, De Geijn R.V (1998) Analysis of a Class of Parallel Matrix Multiplication Algorithms. http://www.cs.utexas.edu/users/plapack/papers/ipps98/ipps98.html, 1998.

Gupta A, Karypis G, Kumar V (1997) IEEE Trans. Parallel Distrib. Syst. 8(5): 502-520, 1997.

Gupta M, Pathak A, Chakrabarti S (2008) Fast algorithms for Top-k Personalized PageRank Queries. In WWW'08. 1225-1226. 2008.

Haveliwala T.H (2002) Topic-sensitive PageRank. WWW'02. 517-526. 2002.

Huang S, Li X, Candan K S, Sapino M L. (2014) "Can you really trust that seed?": Reducing the Impact of Seed Noise in Personalized PageRank. International Conference on Advances in Social Network Analysis and Mining (ASONAM). Beijing, China. 2014

Jeh G, Widom J (2002) Scaling personalized web search. Stanford University Technical Report. 2002.

Kamvar S.D, Haveliwala T.H, Manning C.D, Golub G.H (2003) Extrapolation methods for accelerating PageRank computations. In WWW'03 261-270. 2003.

Karypis G and Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, 1998.

Kim J.H, Candan K.S, Sapino M.L (2013) LR-PPR: Locality-Sensitive, Re-use Promoting, Approximate Personalized PageRank Computation. ACM International Conference on Information and Knowledge Management (CIKM'13), October 2013.

Kim J.H, Candan K.S, Sapino M.L (2012) Impact neighborhood indexing (INI) in diffusion graphs. CIKM'12. 2184-2188, 2012.

Kleinberg J (1999) Authoritative sources in a hyperlinked environment. Journal of the ACM 46 (5): 604632. 1999.

Liu W, Li G, Cheng J. (2013) Fast PageRank approximation by adaptive sampling. Journal of Knowledge and Information Systems (KAIS), 2013.

Lofgren P., Banerjee S., Goel A., Seshadhri C., Fast-PPR: Scaling Personalized PageRank Estimation for Large Graphs. in Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14), pp. 1436-1445, 2014.

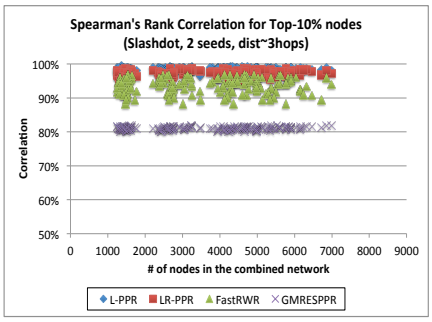
Maehara T, Akiba T, Iwata Y, Kawarabayashi K (2014) Computing Personalized PageRank Quickly by Exploiting Graph Structures. VLDB'14, 2014.

Malewicz G, Austern M, Bik A, Dehnert J, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. SIGMOD'10, 2010.

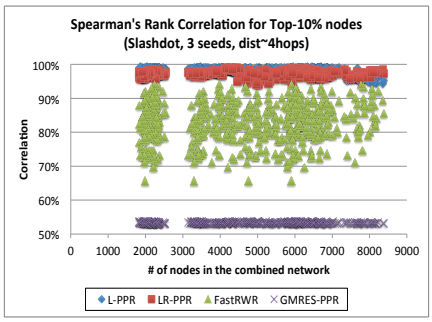
Mei Q, Zhou D, Church K (2008) Query suggestion using hitting time, CIKM'08, 2008.

Newman M, Finding community structure in networks using the eigenvectors of matrices, Phys. Rev. E 74, 036104, 2006.

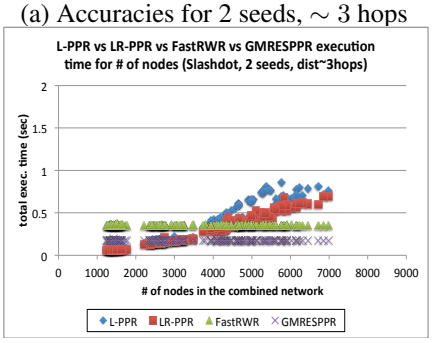
Palmer C, Gibbons P, Faloutsos C (2002) Anf: a fast and scalable tool for data mining in massive graphs. KDD'02, 2002.
Pease Marshall C (1967) Matrix Inversion Using Parallel. Processing. J. ACM 14, 4, 757-764, Oct. 1967.
Piegorsch W, Casella G.E (1990) Inverting a sum of matrices. In SIAM Review, 1990.
Sarkar P, Moore A.W, Prakash A (2008) Fast incremental proximity search in large graphs. ICML'08, 2008.
Song H.H, Cho T.W, Dave V, Zhang Y, Qiu L (2009) Scalable proximity estimation and link prediction in online social networks. In *Internet Measurement Conference*, pp. 322–335. ACM, 2009.
Tong H, Faloutsos C (Center-piece subgraphs: problem definition and fast solutions. In *KDD* pp. 404–413, 2006.
Tong H, Faloutsos C, Koren Y (2007) Fast direction-aware proximity for graph mining. *KDD*, pp. 747–756, 2007.
Tong H, Faloutsos C, Pan J.Y (2006) Fast Random Walk with Restart and Its Applications. In ICDM'06. 613-622. 2006.
Wei F (2010) Tedi: efficient shortest path query answering on graphs. SIGMOD'10.
Williams V.V (2011) Breaking the Coppersmith-Winograd barrier. Unpublished manuscript. http://www.cs.berkeley.edu/ virgi/matrixmult.pdf, 2011.
Wu Y, Raschid L (2009) ApproxRank: Estimating Rank for a Subgraph, ICDE'09, 54-65, 2009.
Xiao Y, Wu W, Pei J, Wang W, He Z (2009) Efficiently indexing shortest paths by exploiting symmetry in graphs. EDBT, 2009.
Zhou L, Chen L, Ozsu M.T (2009) Distance-join: pattern match query in a large graph, VLDB, 2009.



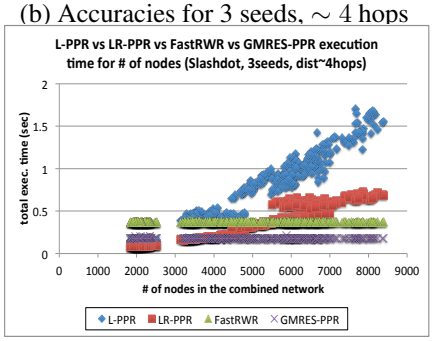
(a) Accuracies for 2 seeds, $\sim$ 3 hops



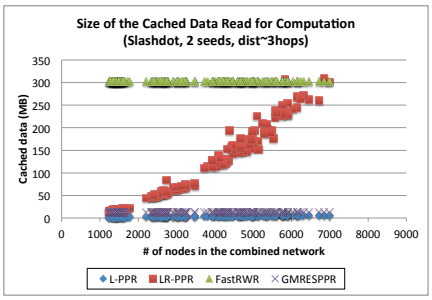
(b) Accuracies for 3 seeds, $\sim$ 4 hops



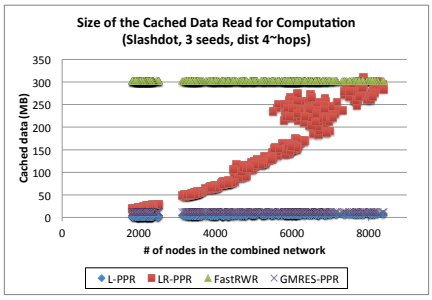
(c) Exec. times for 2 seeds, $\sim$ 3 hops



(d) Exec. times for 3 seeds, $\sim$ 4 hops



(e) Data for 2 seeds, $\sim$ 3 hops



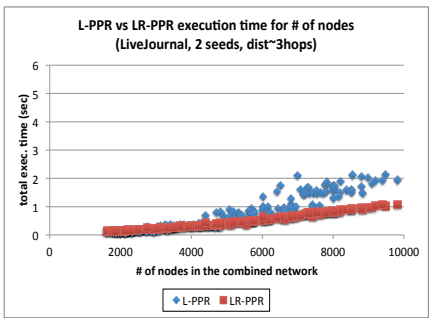
(f) Data for 3 seeds, $\sim$ 4 hops

Fig. 19. Performances of L-PPR, LR-PPR, and FastRWR as a function of the size of the combined localities network (SlashDot data set) for different numbers of seeds, selected at varying hop distances from each other

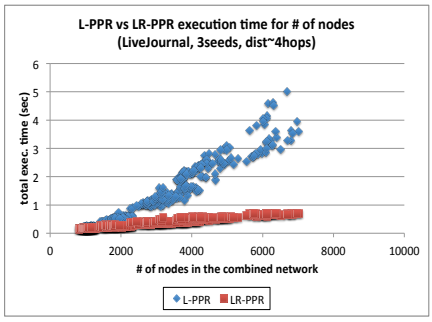
Appendix

• *SlashDot Data Set:* Figure 19 compares in further detail the execution times, accuracies, and amounts of data read by L-PPR, LR-PPR, FastRWR, GMRES-PPR from the cache per query as a function of the size of the merged locality network for different seeds and target locality sizes of the SlashDot data set. Since the SlashDot and Epinions Data sets are similar (Table 1), the results in Figure 19 are also similar to the results for the Epinions data set presented in Section 5.3.1, Figure 11.

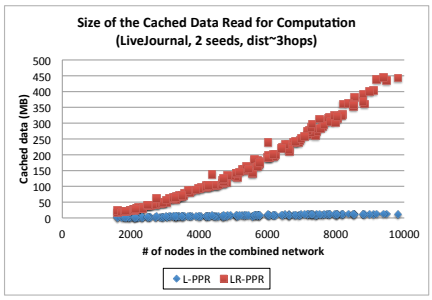
• *LiveJournal Data Set:* Figure 20 also shows and compares the execution times and amounts of data read by L-PPR and LR-PPR from the cache per query as a function



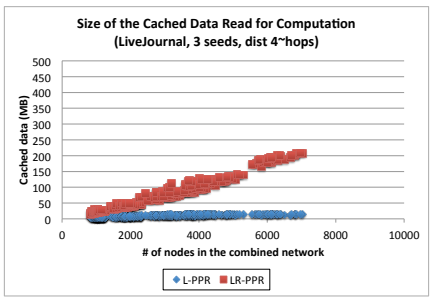
(a) Exec. times for 2 seeds, $\sim$ 3 hops



(b) Exec. times for 3 seeds, $\sim$ 4 hops



(c) Data for 2 seeds, $\sim$ 3 hops



(d) Data for 3 seeds, $\sim$ 4 hops

Fig. 20. Performances of L-PPR, LR-PPR, and FastRWR as a function of the size of the combined localities network (LiveJournal data set) for different numbers of seeds, selected at varying hop distances from each other

of the size of the merged locality network for different seeds and target locality sizes of the LiveJournal data set. Note that we could not compare the accuracies because we could not compute Global PPR. The difference from other datasets is that the number of nodes in 3 hops is larger than the number of nodes in 4 hops. The results show that it follows the same pattern as other data sets' results on the execution time and the size of cached data.

Author Biographies

Jung Hyun Kim received a BS degree in information and communication engineering from Sungkyunkwan university, South Korea in 2003 and a MS degree from Illinois Institute of Technology, USA in 2005, where he did research works in the database group. He is currently a Ph.D. student at the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, USA. His research interests include information propagation in social network, large-scale data mining, information retrieval, and recommender systems.

K. Selçuk Candan is a Professor of Computer Science and Engineering at the Arizona State University. His research interests include scalable data management of large, heterogeneous data sets. He has published over journal and peer-reviewed conference articles, one book, and 16 book chapters. He has 9 patents. Prof. Candan served as an associate editor of the Very Large Databases (VLDB) journal. He currently serves as associate editor for the ACM Transactions on Database Systems, IEEE Transactions on Multimedia, and the Journal of Multimedia. He has served in the organization and program committees of various conferences. He has successfully served as the PI or co-PI of numerous grants, including from the National Science Foundation, Air Force Office of Research, Army Research Office, Mellon Foundation, and HP Labs. He is a member of the Executive Committee of ACM SIGMOD and an ACM Distinguished Scientist.

Maria Luisa Sapino is a Professor of Computer Science at the University of Torino. She leads the Heterogeneous Data Management (HDM) group at UNITO and has published extensively in data management, multimedia, and social media analysis venues, including a recent textbook on "Data Management for Multimedia Retrieval," published by the Cambridge University Press. She served as the chair for the International Workshop on Ambient Media Delivery and Interactive Television (AMDIT08), as general chair for the KDD Workshop on Multimedia Data Mining 2008, as a program co-chair for the SIAM Workshop on Multimedia Data Mining 2009. She served as a workshops chair for ACM Multimedia 2011, a publicity chair for ACM SIGMOD 2012, and a workshops chair for the IEEE International Conference on Multimedia and Expo (ICME) 2013. Her past and current industrial collaborations in the areas of digital media and smart television include RAI, Telefonica Madrid, and Telecom Italia.

Correspondence and offprint requests to: Jung Hyun Kim, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA. Email: jkim294@asu.edu