IRWR: Incremental Random Walk with Restart

Weiren Yu[†‡], Xuemin Lin[♮†]
[†]The University of New South Wales, Australia [‡]NICTA, Australia
[♮]East China Normal University, China
{weirenyu, lxue}@cse.unsw.edu.au

ABSTRACT

Random Walk with Restart (RWR) has become an appealing measure of node proximities in emerging applications *e.g.,* recommender systems and automatic image captioning. In practice, a real graph is typically large, and is frequently updated with small changes. It is often cost-inhibitive to recompute proximities from scratch via *batch* algorithms when the graph is updated. This paper focuses on the incremental computations of RWR in a dynamic graph, whose edges often change over time. The prior attempt of RWR [1] deploys k-dash to find top-k highest proximity nodes for a given query, which involves a strategy to incrementally *estimate* upper proximity bounds. However, due to its aim to prune needless calculation, such an incremental strategy is *approximate*: in $O(1)$ time for each node. The main contribution of this paper is to devise an *exact* and fast incremental algorithm of RWR for edge updates. Our solution, IRWR, can incrementally compute any node proximity in $O(1)$ time for each edge update without loss of exactness. The empirical evaluations show the high efficiency and exactness of IRWR for computing proximities on dynamic networks against its batch counterparts.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Storage and Retrieval

Keywords

Random Walk with Restart; Proximity; Dynamic graph

1. INTRODUCTION

Measuring node proximities in graphs is a key task of web search. Due to various applications in recommender systems and social networks, many proximity metrics have come into play. For instance, Brin and Page [2] invented PageRank to determine the ranking of web pages. Jeh and Widom [3] proposed SimRank to assess node-to-node proximities. Random Walk with Restart (RWR) [4] is one of such useful proximity metrics for ranking nodes in order of relevance to a query node. In RWR, the proximity of node u *w.r.t.* query node q is defined as the limiting probability that a random surfer, starting from q, and then iteratively either moving to one of its out-neighbors with probability weighted by the edge weights, or restarting from q with probability c, will eventually arrive at node u. Recently, RWR has received increasing attention (*e.g.,* for collaborative filtering [1] and image labeling [5]) since it can fairly capture the global structure of graphs, and relations in interlinked networks [6].

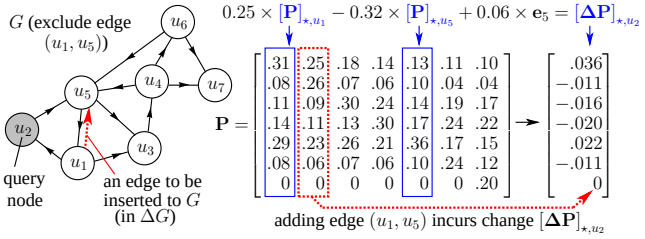


Figure 1: Computing RWR Incrementally

Prior RWR computing methods are based on static graphs, which is costly: Given a graph $G(V, E)$, and a query $q \in V$, k-dash [1] yields, in the worse case, $O(|V|^2)$ time and space, which, in practice, can be bounded by $O(|E| + |V|)$, to find top-k highest proximity nodes. B_LIN and NB_LIN [4] need $O(|V|^2)$ time and space for computing all node proximities. In general, real graphs are often constantly updated with small changes. This calls for the need for incremental algorithms to compute proximities. We state the problem below:

Problem (Incremental Update for RWR)
Given a graph G, proximities $\mathbf{P}$ for G, changes ΔG to G, a query node q, and a restarting probability $c \in (0, 1)$.
Compute changes to the proximities *w.r.t.* q exactly.

Here, $\mathbf{P}$ is a proximity matrix whose entry $[\mathbf{P}]_{i,j}$ denotes the proximity of node i *w.r.t.* query j, and ΔG is comprised of a set of edges to be inserted into or deleted from G.

In contrast with the existing *batch* algorithms [1, 4] that recompute the updated proximities from scratch, our incremental algorithm can exploit the dynamic nature of graphs by pre-computing proximities only once on the entire graph via a batch algorithm, and then *incrementally* computing their changes in response to updates. The response time of RWR can be greatly improved by maximal use of previous computation, as shown in Example 1.

Example 1. *Figure 1 depicts a graph G, taken from [1]. Given the query u_2, the old proximities $\mathbf{P}$ for G, and $c = 0.2$, we want to compute new proximities w.r.t. u_2 when there is an edge (u_1, u_5) inserted into G, denoted by ΔG. The existing methods, k-dash and B_LIN, have to recompute the new proximities in $G \cup \Delta G$ from scratch, without using the previously computed proximities in G, which is costly.*

However, we observe that the increment $[\mathbf{\Delta P}]_{\star,u_2}$ [1] *to the old* $[\mathbf{P}]_{\star,u_2}$ *is the linear combination of* $[\mathbf{P}]_{\star,u_1}$ *and* $[\mathbf{P}]_{\star,u_5}$*,i.e.,*

$$[\mathbf{\Delta P}]_{\star,u_2} = \alpha \cdot [\mathbf{P}]_{\star,u_1} + \beta \cdot [\mathbf{P}]_{\star,u_5} + \lambda \cdot \mathbf{e}_5 \text{ [2]} \quad (1)$$

with $\alpha = 0.25,\ \beta = -0.32,\ \lambda = 0.06$. *Hence, there are opportunities to incrementally compute the changes* $[\mathbf{\Delta P}]_{\star,u_2}$ *by fully utilizing the old proximities of* $\mathbf{P}$. *As opposed to k-dash and B_LIN involving* matrix-vector multiplications*, computing* $[\mathbf{\Delta P}]_{\star,u_2}$ *via Eq.*(1) *only needs* vector scaling and additions*, thus greatly improving the response time.* □

As suggested by Example 1, when the graph G is updated, it is imperative to incrementally compute new proximities by leveraging information from the old proximities. However, it is a grand challenge to characterize the changes $[\mathbf{\Delta P}]_{\star,q}$ in terms of a linear combination of the columns in old $\mathbf{P}$, since it seems hard to determine the scalars α, β, λ for Eq.(1). Worse still, much less is known about how to extract a subset of columns from the old $\mathbf{P}$ (*e.g.,* why $[\mathbf{P}]_{\star,u_1}$ and $[\mathbf{P}]_{\star,u_5}$ are chosen from $\mathbf{P}$ in Eq.(1)), to express the changes $[\mathbf{\Delta P}]_{\star,q}$.

Contributions. This paper aims to tackle these problems. To the best of our knowledge, this work makes the first effort to study incremental RWR computing in evolving graphs, with no loss of exactness. 1) We first consider *unit update*, *i.e.,* a single-edge insertion or deletion, and derive an elegant formula that characterizes the proximity changes as a linear combination of the columns from the old proximity matrix. 2) We then devise an incremental algorithm for *batch update*, *i.e.,* a list of edge deletions and insertions mixed together, and show that any node proximity can be computed in $O(1)$ time for every edge update, with no sacrifice in accuracy. 3) Our empirical study demonstrates that the incremental approach greatly outperforms *k*-dash [1], a batch algorithm that is reported as the best for RWR proximity computing, when networks are constantly updated.

Organization. Section 2 overviews the background of RWR. The incremental RWR is studied in Section 3. Section 4 gives empirical results, followed by open issues in Section 5.

Related Work. Incremental algorithms have proved useful in various node proximity computations on evolving graphs, such as the personalized PageRank [7] and SimRank [8]. However, very few results are known on incremental RWR computing, far less than their batch counterparts [1, 4, 9]. *k*-dash [1] is the best known approach to finding top-k highest RWR proximity nodes for a given query, which involves a strategy to incrementally *estimate* upper proximity bounds. Nevertheless, such an incremental strategy is *approximate*: in $O(1)$ time for each node, which is mainly developed for pruning unnecessary computation. In contrast, our incremental algorithm can, without loss of exactness, compute any node proximity in $O(1)$ time for every edge update. Moore *et al.* [10] leveraged a sampling approach with branch and bound pruning to find near neighbors of a query *w.h.p.*. However, their incremental algorithm is *probabilistic*. Later, Zhou *et al.* [9] generalized the original RWR by incorporating node attributes into link structure for graph clustering. Based on this, an incremental version of [9] was proposed by Cheng *et al.* [11], with the focus to support *attribute update*. It differs from this work in that our incremental algorithm is designed for *structure update*. Thus, [11] cannot cope with *hyperlink changes* incrementally in dynamic graphs.

[1] $[\mathbf{X}]_{\star,j}$ denotes the j-th column vector of matrix $\mathbf{X}$.

[2] $\mathbf{e}_i$ is the $|V| \times 1$ unit vector with a 1 in the i-th entry.

2. PRELIMINARIES

We formally overview the background of this paper. Graphs studies here are directed graphs with no multiple edges.

RWR Formula [1]. In a graph $G(V, E)$, let $\mathbf{A}$ be the transition matrix (*i.e.,* column normalized adjacency matrix) of G, whose entry $[\mathbf{A}]_{u,v} = \frac{1}{d_v}$ if $(u, v) \in E$, and 0 otherwise. Here, d_v denotes the in-degree of v. Given query node $q \in V$, and restart probability $c \in (0, 1)$, the proximity of node u *w.r.t.* q, denoted by $[\mathbf{P}]_{u,q}$, is recursively defined as follows:

$$[\mathbf{P}]_{\star,q} = (1 - c) \cdot \mathbf{A} \cdot [\mathbf{P}]_{\star,q} + c \cdot \mathbf{e}_q \quad (2)$$

where $[\mathbf{P}]_{\star,q}$ is the $|V| \times 1$ *proximity vector w.r.t.* q (*i.e.,* the q-th column of matrix $\mathbf{P}$), whose u-th entry equals to $[\mathbf{P}]_{u,q}$. $\mathbf{e}_q$ is the $|V| \times 1$ unit *query vector*, whose q-th entry is 1.

Intuitively, $[\mathbf{P}]_{u,q}$ is the limiting probability, denoting the long-term visit rate of node u, given a bias toward query q.

The RWR proximity defined in Eq.(2) can be rewritten as

$$[\mathbf{P}]_{\star,q} = c(\mathbf{I} - (1 - c) \cdot \mathbf{A})^{-1} \cdot \mathbf{e}_q \quad (3)$$

where $\mathbf{I}$ is the $|V| \times |V|$ identity matrix.

Existing methods of computing RWR are in a batch style, with the aim to accelerate the matrix inversion in Eq.(3). For instance, *k*-dash [1] uses LU decomposition and an incremental pruning strategy to speed up the matrix inversion.

3. INCREMENTAL RWR COMPUTING

We now study the incremental RWR computation. Given the old $\mathbf{P}$ for G, changes ΔG to G, query q, and $c \in (0, 1)$, the goal is to compute $[\mathbf{\Delta P}]_{\star,q}$ for ΔG. The key idea of our approach is to maximally reuse the previous computation, by characterizing $[\mathbf{\Delta P}]_{\star,q}$ as a linear combination of the columns from the old $\mathbf{P}$. The main result is as follows.

THEOREM 1. *Any node proximity of a given query can be incrementally computed in* $O(1)$ *time for each edge update.*

To prove Theorem 1, we first consider unit edge update, and then devise an incremental algorithm for batch updates, with the desired complexity bound.

Unit Update. The update (insertion/deletion) of an edge from G may lead to the changes $[\mathbf{\Delta P}]_{\star,q}$ of the proximity. We incrementally compute $[\mathbf{\Delta P}]_{\star,q}$ based on the following.

PROPOSITION 1. *Given a query* q*, and the old proximity matrix* $\mathbf{P}$ *for* G*, if there is an edge insertion* (i, j) *into* G*, then the changes* $[\mathbf{\Delta P}]_{\star,q}$ *w.r.t.* q *can be computed as*

$$[\mathbf{\Delta P}]_{\star,q} = \frac{(1-c)[\mathbf{P}]_{j,q}}{1-(1-c)[\mathbf{y}]_j} \cdot \mathbf{y} \text{ with} \quad (4)$$

$$\mathbf{y} = \begin{cases} \frac{1}{c}[\mathbf{P}]_{\star,i} & (d_j = 0) \\ \frac{1}{c(d_j+1)}([\mathbf{P}]_{\star,i} - \frac{1}{1-c}[\mathbf{P}]_{\star,j}) + \frac{1}{(1-c)(d_j+1)}\mathbf{e}_j & (d_j > 0) \end{cases}$$

where d_j *is the in-degree of node* j *in the old* G*, and* $[\mathbf{y}]_j$ *is the* j*-th entry of vector* $\mathbf{y}$.

If there is an edge deletion (i, j) *from* G*, then* $[\mathbf{\Delta P}]_{\star,q}$ *can also be computed via Eq.*(4) *with* $\mathbf{y}$ *being replaced by*

$$\mathbf{y} = \begin{cases} -\frac{1}{c}[\mathbf{P}]_{\star,i} & (d_j = 1) \\ \frac{1}{c(d_j-1)}(\frac{1}{1-c}[\mathbf{P}]_{\star,j} - [\mathbf{P}]_{\star,i}) - \frac{1}{(1-c)(d_j-1)}\mathbf{e}_j & (d_j > 1) \end{cases}$$

As opposed to the traditional methods, *e.g., k*-dash and B_LIN, that requires *matrix-vector multiplications* to compute new proximities via Eq.(2), Proposition 1 allows merely *vector scaling and additions* for efficiently computing $[\mathbf{\Delta P}]_{\star,q}$.

The proof of Proposition 1 is attained by combining the three following lemmas.

LEMMA 1. *Let* $\mathbf{A}$ *be the old transition matrix of* G. *If there is an edge insertion* (i,j) *into* G, *then the new transition matrix* $\tilde{\mathbf{A}}$ *is updated by*

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{a}\mathbf{e}_j^T \text{ with } \mathbf{a} = \begin{cases} \mathbf{e}_i & (d_j = 0) \\ \frac{1}{d_j+1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j}) & (d_j > 0) \end{cases} \quad (5)$$

If there is an edge deletion (i,j) *from* G, *then the new* $\tilde{\mathbf{A}}$ *is also updated as Eq.*(5) *with* $\mathbf{a}$ *being replaced by*

$$\mathbf{a} = \begin{cases} \mathbf{e}_i & (d_j = 1) \\ \frac{1}{d_j-1}([\mathbf{A}]_{\star,j} - \mathbf{e}_i) & (d_j > 1) \end{cases} \quad (6)$$

PROOF. Due to space limits, we shall merely prove the insertion case. A similar proof holds for the deletion case.

(i) When $d_j = 0$, $[\mathbf{A}]_{\star,j} = \mathbf{0}$. Thus, for an inserted edge (i,j), $[\mathbf{A}]_{i,j}$ will be updated from 0 to 1, *i.e.,* $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{e}_i\mathbf{e}_j^T$.

(ii) When $d_j > 0$, all the nonzero entries of $[\mathbf{A}]_{\star,j}$ are $\frac{1}{d_j}$. Thus, for an inserted edge (i,j), we first update $[\mathbf{A}]_{i,j}$ from 0 to $\frac{1}{d_j}$, *i.e.,* $\mathbf{A} \Rightarrow \mathbf{A} + \frac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T$, and then change all nonzero entries of $[\mathbf{A} + \frac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T]_{\star,j}$ from $\frac{1}{d_j}$ to $\frac{1}{d_j+1}$. Recall from the elementary matrix property that multiplying the j-th column of a matrix by $\alpha \neq 0$ can be accomplished by using $\mathbf{I} - (1-\alpha)\mathbf{e}_j\mathbf{e}_j^T$ as a right-hand multiplier on the matrix. Hence, scaling $[\mathbf{A} + \frac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T]_{\star,j}$ by $\alpha = \frac{d_j}{d_j+1}$ yields

$$\begin{aligned} \tilde{\mathbf{A}} &= (\mathbf{A} + \tfrac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T)(\mathbf{I} - (1 - \tfrac{d_j}{d_j+1})\mathbf{e}_j\mathbf{e}_j^T) \\ &= \mathbf{A} + \tfrac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T - \tfrac{1}{d_j+1}(\mathbf{A} + \tfrac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T)\mathbf{e}_j\mathbf{e}_j^T \\ &= \mathbf{A} + \tfrac{1}{d_j+1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j})\mathbf{e}_j^T \end{aligned}$$

Combining (i) and (ii), Eq.(5) is derived. □

Lemma 1 suggests that each edge change will incur a rank-one update of $\mathbf{A}$. To see how the update of $\mathbf{A}$ affects the changes to $\mathbf{P}$, we have the following lemma.

LEMMA 2. *Let* $\mathbf{P}$ *be the old proximity matrix for* G. *If there is an edge update* (i,j) *to* G, *then the new proximity* $\tilde{\mathbf{P}}$ *w.r.t. a given query* q *is updated as*

$$[\tilde{\mathbf{P}}]_{\star,q} = [\mathbf{P}]_{\star,q} + (1-c)\gamma \cdot \mathbf{z} \quad (7)$$

$$\text{with } \gamma = \frac{[\mathbf{P}]_{j,q}}{1-(1-c)\cdot[\mathbf{z}]_j} \text{ and } \mathbf{z} = (\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a},$$

where the vector $\mathbf{a}$ *is defined by Lemma 1.*

PROOF. By RWR definition in Eq.(3), $[\tilde{\mathbf{P}}]_{\star,q}$ satisfies

$$(\mathbf{I} - (1-c)\tilde{\mathbf{A}}) \cdot [\tilde{\mathbf{P}}]_{\star,q} = c\mathbf{e}_q \quad (8)$$

where $\tilde{\mathbf{A}}$ is the new transition matrix that is expressed as $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{a}\mathbf{e}_j^T$ by Lemma 1. Thus, Eq.(8) is rewritten as

$$\begin{bmatrix} \mathbf{I} - (1-c)\mathbf{A} & -(1-c)\mathbf{a} \\ \mathbf{e}_j^T & -1 \end{bmatrix} \begin{bmatrix} [\tilde{\mathbf{P}}]_{\star,q} \\ \gamma \end{bmatrix} = \begin{bmatrix} c\mathbf{e}_q \\ 0 \end{bmatrix} \quad (9)$$

To solve $[\tilde{\mathbf{P}}]_{\star,q}$ and γ in Eq.(9), we apply block elimination, by using block elementary row operations, and starting with the associated augmented matrix:

$$\left[\begin{array}{cc|c} \mathbf{I} - (1-c)\mathbf{A} & -(1-c)\mathbf{a} & c\mathbf{e}_q \\ \mathbf{e}_j^T & -1 & 0 \end{array}\right] \xrightarrow{\text{Row2} - \mathbf{e}_j^T(\mathbf{I}-(1-c)\mathbf{A})^{-1}\cdot\text{Row1}}$$

$$\rightarrow \left[\begin{array}{cc|c} \mathbf{I} - (1-c)\mathbf{A} & -(1-c)\mathbf{a} & c\mathbf{e}_q \\ 0 & (1-c)\mathbf{e}_j^T(\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a} - 1 & -c\mathbf{e}_j^T[\mathbf{P}]_{\star,q} \end{array}\right]$$

The final array represents the following equations:

$$\begin{cases} (\mathbf{I} - (1-c)\mathbf{A})[\tilde{\mathbf{P}}]_{\star,q} - \gamma(1-c)\mathbf{a} = c\mathbf{e}_q \\ \left((1-c)\mathbf{e}_j^T(\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a} - 1\right)\gamma = -c[\mathbf{P}]_{j,q} \end{cases}$$

Back substitution, along with Eq.(3), yields Eq.(7). □

Algorithm 1: IRWR $(G, \mathbf{P}, q, \Delta G, c)$

Input : graph G, old proximities $\mathbf{P}$ for G, query node q, updates ΔG to G, and restarting probability c.
Output: new proximities $[\tilde{\mathbf{P}}]_{\star,q}$ *w.r.t.* q.

```
foreach edge (i,j) ∈ ΔG to be updated do
    d_j := in-degree of node j in G ;
    if edge (i,j) is to be inserted then
        if d_j = 0 then y := (1/c)[P]_{⋆,i} ;
        else y := 1/(d_j+1) ((1/c)([P]_{⋆,i} − 1/(1−c)[P]_{⋆,j}) + 1/(1−c) e_j) ;
        G := G ∪ {(i,j)} ;
    else if edge (i,j) is to be deleted then
        if d_j = 1 then y := −(1/c)[P]_{⋆,i} ;
        else y := 1/(d_j−1) ((1/c)(1/(1−c)[P]_{⋆,j} − [P]_{⋆,i}) − 1/(1−c) e_j) ;
        G := G\{(i,j)} ;
    ΔG := ΔG\{(i,j)} ;
    if ΔG ≠ ∅ then
        foreach v ∈ {vertices in ΔG} ∪ {q} do
            γ := [P]_{j,v} / (1−(1−c)[y]_j),  [P]_{⋆,v} := [P]_{⋆,v} + (1−c)γy ;
    else γ := [P]_{j,q} / (1−(1−c)[y]_j),  [P̃]_{⋆,q} := [P]_{⋆,q} + (1−c)γy ;
return [P̃]_{⋆,q} ;
```

Lemma 2 tells that for each edge update, the changes to $\mathbf{P}$ are just associated with the scaling operation of vector $\mathbf{z}$. However, it is costly to compute $\mathbf{z}$ via Eq.(7) as it involves the inversion of a matrix. Lemma 3 provides an efficient way of computing $(\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a}$ from a few columns of $\mathbf{P}$.

LEMMA 3. *Suppose there is an edge update* (i,j) *to* G, *and* $\mathbf{a}$ *is defined by Lemma 1. Then,* $(\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a} = \mathbf{y}$ *with* $\mathbf{y}$ *being defined in Proposition 1.*

PROOF. Due to space limits, we shall only prove the edge insertion case. A similar proof holds for the deletion case.

(i) When $d_j = 0$, $\mathbf{a} = \mathbf{e}_i$. Then, Eq.(3) implies that

$$(\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{e}_i = \tfrac{1}{c}[\mathbf{P}]_{\star,i}$$

(ii) When $d_j > 0$, $\mathbf{a} = \frac{1}{d_j+1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j})$. Then,

$$\begin{aligned} (\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a} &= \tfrac{1}{d_j+1}(\mathbf{I} - (1-c)\mathbf{A})^{-1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j}) \\ &= \tfrac{1}{d_j+1}(\tfrac{1}{c}[\mathbf{P}]_{\star,i} - (\mathbf{I} - (1-c)\mathbf{A})^{-1}[\mathbf{A}]_{\star,j}) \end{aligned}$$

To solve $(\mathbf{I} - (1-c)\mathbf{A})^{-1}[\mathbf{A}]_{\star,j}$, we apply the property that $(\mathbf{I} - \mathbf{X})^{-1} = \sum_{k=0}^{\infty}\mathbf{X}^k$ (for $\|\mathbf{X}\|_1 < 1$) and obtain

$$\begin{aligned} (\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{A} &= \sum_{k=0}^{\infty}(1-c)^k\mathbf{A}^{k+1} = \tfrac{1}{1-c}\sum_{k=1}^{\infty}(1-c)^k\mathbf{A}^k \\ &= \tfrac{1}{1-c}((\mathbf{I} - (1-c)\mathbf{A})^{-1} - \mathbf{I}) = \tfrac{1}{1-c}(\tfrac{1}{c}\mathbf{P} - \mathbf{I}) \end{aligned}$$

Thus, we have $(\mathbf{I} - (1-c)\mathbf{A})^{-1}[\mathbf{A}]_{\star,j} = \frac{1}{1-c}(\frac{1}{c}[\mathbf{P}]_{\star,j} - \mathbf{e}_j)$. Substituting this back produces the final results. □

Combining Lemmas 1–3 together proves Proposition 1.

Algorithm for Batch Updates. Based on Proposition 1, we devise IRWR, an incremental RWR algorithm to handle a set ΔG of edge insertions and deletions (batch update).

IRWR is shown in Algorithm 1. Given the old $\mathbf{P}$ for G *w.r.t.* query q, and the batch edge updates ΔG, it computes new proximities *w.r.t.* q in $G \cup \Delta G$ without loss of exactness.

It works as follows. For each edge (i, j) to be updated, it first computes the auxiliary vector $\mathbf{y}$ from a linear combination of only a few columns in $\mathbf{P}$ (lines 2–10). Using $\mathbf{y}$, it then (i) removes (i, j) from ΔG (line 11) and (ii) updates the proximities *w.r.t.* each remaining node in ΔG (lines 12–14). After all the edges are eliminated from ΔG, IRWR finally calculates the new proximities $[\tilde{\mathbf{P}}]_{\star,q}$ from $\mathbf{y}$ (line 15).

EXAMPLE 2. *Recall* $\mathbf{P}$ *and* G *of Figure 1. Consider batch updates* ΔG*, which insert edge* (u_1, u_5) *and delete* (u_4, u_6)*, where* (u_1, u_5) *is given in Example 1. IRWR computes the new proximities* $[\mathbf{P}]_{\star,u_2}$ *w.r.t. query* u_2 *in* $G+\Delta G$ *as follows:*

For the edge insertion (u_1, u_5)*, since* $d_{u_5} = 3$ *and* $c = 0.2$*,* $\mathbf{y} = 1.25 \times [\mathbf{P}]_{\star,u_1} - 1.56 \times [\mathbf{P}]_{\star,u_5} + 0.31 \times \mathbf{e}_5$ *(via line 5).*

Before proceeding with the edge deletion, let us look at the changes $[\mathbf{\Delta P}]_{\star,u_2}$ *(via line 14) for the inserted* (u_1, u_5)*:*

$$[\mathbf{\Delta P}]_{\star,u_2} = (1-c)\gamma \cdot \mathbf{y} \quad \text{with } \gamma = \frac{[\mathbf{P}]_{u_5,u_2}}{1-(1-c)[\mathbf{y}]_5} = 0.254$$
$$= 0.25 \times [\mathbf{P}]_{\star,u_1} - 0.32 \times [\mathbf{P}]_{\star,u_5} + 0.06 \times \mathbf{e}_5,$$

which explains why the values of α, β, λ *are chosen for E-q.*(1)*.*

IRWR then removes (u_1, u_5) *from* ΔG *(line 11). Using* $\mathbf{y}$*, it updates proximities w.r.t.* $u_4, u_6 \in \Delta G$ *(lines 12–14). Thus,* $[\mathbf{P}]_{\star,u_4} = (.17, .05, .23, .28, .23, .05, 0)^T$*, and* $[\mathbf{P}]_{\star,u_6} = (.14, .04, .18, .23, .18, .24, 0)^T$ *after* $(u1, u5)$ *is added to* G*.*

Likewise, for the edge deletion (u_4, u_6)*,* $d_{u_6} = 1$ *implies* $\mathbf{y} = -\frac{1}{0.2} \times [\mathbf{P}]_{\star,u_4}$ *(line 8). Then,* (u_4, u_6) *is removed from* ΔG *(line 11). Since* $\Delta G = \varnothing$*, the changes* $[\mathbf{\Delta P}]_{\star,u_2}$ *for the deleted* (u_4, u_6) *is obtained (via line 15):*

$$[\mathbf{\Delta P}]_{\star,u_2} = 0.8\gamma \cdot \mathbf{y} = -0.17 \times [\mathbf{P}]_{\star,u_4} \text{ with } \gamma = \frac{[\mathbf{P}]_{u_6,u_2}}{1-(1-c)[\mathbf{y}]_6} = 0.04$$
$$\Rightarrow [\tilde{\mathbf{P}}]_{\star,u_2} = [\mathbf{P}]_{\star,u_2} + [\mathbf{\Delta P}]_{\star,u_2} = (.25, .24, .04, .04, .22, .04, 0)^T. \quad \square$$

Correctness & Complexity. To complete the proof of Theorem 1, we notice that (i) IRWR can correctly compute RWR proximities, which is verified by Proposition 1. Moreover, IRWR always terminates, since the size of ΔG is monotonically decreasing. (ii) One can readily verify that for each edge update, IRWR involves only vector scaling and additions, which is in $O(1)$ time for each node proximity.

4. EXPERIMENTAL EVALUATION

We present an empirical study on real and synthetic data to evaluate the efficiency of IRWR for incremental computation, as compared with (a) its batch counterpart B_LIN [4], (b) k-dash [1], the best known algorithm for top-k search, and (c) IncPPR [7], the incremental personalized PageRank.

Two real datasets are adopted: (a) p2p-Gnutella, a Gnutella P2P digraph, in which nodes represent hosts, and edges host connections. The dataset has 62.5K nodes and 147.9K edges. (b) cit-HepPh, a citation network from Arxiv, where nodes denote papers, and edges paper citations. We extracted a snapshot with 27.7K nodes and 352.8K edges.

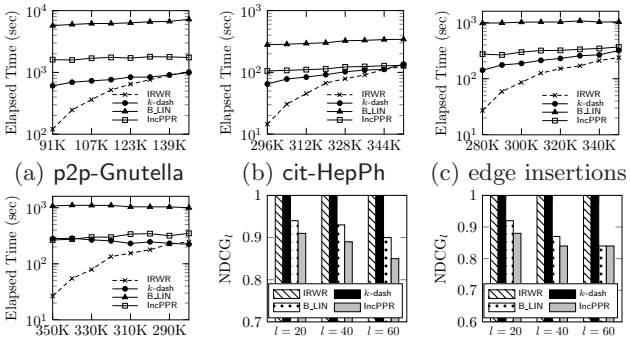
GraphGen[3] is used to build synthetic graphs and updates. Graphs are controlled by (a) the number of nodes $|V|$, and (b) the number of edges $|E|$; updates by (a) update type (edge insertion or deletion), and (b) the size of updates $|\Delta G|$.

All the algorithms are implemented in Visual C++ 10.0. We used a machine with an Intel Core(TM) 3.10GHz CPU and 8GB RAM, running Windows 7.

We set restarting probability $c = 0.2$ in our experiments.

Experimental Results. We next present our findings.

[3]http://www.cse.ust.hk/graphgen/



(a) p2p-Gnutella (b) cit-HepPh (c) edge insertions

(d) edge deletions (e) p2p-Gnutella (f) cit-HepPh

Figure 2: Performance Evaluation of IRWR

1) Incremental Efficiency. We first evaluate the computational time of IRWR on both real and synthetic data.

Figures 2(a) and 2(b) depict the results for edges inserted to p2p-Gnutella ($|V|$=62.5K) and cit-HepPh ($|V|$=27.7K), respectively. Fixing $|V|$, we vary $|E|$ as shown in the x-axis. Here, the updates are the difference of snapshots *w.r.t.* the collection time of datasets, reflecting their real-life evolution. We find that (a) IRWR outperforms k-dash on p2p-Gnutella for 92.7% (*resp.* cit-HepPh for 97.5%) of edge updates. When the changes are 61.9% on p2p-Gnutella (83.8% on cit-HepPh), IRWR improves k-dash by over 5.1x (*resp.* 4.4x). This is because IRWR reuses the old information in G for incrementally updating proximities via vector scaling and additions, without the need for expensive LU decomposition of k-dash. (b) IRWR always is better than B_LIN by nearly one order of magnitude as B_LIN requires costly block matrix inversions. (c) IRWR outperforms IncPPR for over 95% of insertions, due to the extra cost of IncPPR for doing short random walks. (d) IRWR is sensitive to $|\Delta G|$ as the larger $|\Delta G|$ is, the larger the affected area is, so is the computation cost, as expected.

Fixing $|V|$=50K on synthetic data, we varied $|E|$ from 280K to 350K (*resp.* from 350K to 280K) in 10K increments (*resp.* decrements). The results are shown in Figures 2(c) and 2(d), respectively, analogous to those on real datasets.

2) Exactness. To measure IRWR accuracy, we adopted NDCG$_l$ (Normalized Discounted Cumulative Gain) for ranking top-l node proximities with $l = 20, 40, 60$, and chose the ranking results of k-dash as the benchmark, due to its exactness. The results on p2p-Gnutella and cit-HepPh are reported in Figures 2(e) and 2(f), indicating that IRWR never scarifies accuracy for achieving high efficiency, superior to other approaches.

5. CONCLUSIONS

We showed how RWR proximities can be computed very efficiently in an incremental update model, where the edges of a graph are constantly changed. We also empirically evaluated that IRWR greatly outperforms the other approaches on both real and synthetic graphs without loss of exactness. Our future work will further predict up to what fraction of updated edges IRWR is faster than its batch counterparts.

6. REFERENCES

[1] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-k search for random walk with restart," *PVLDB*, vol. 5, pp. 442–453, 2012.

[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," tech. rep., Stanford InfoLab, 1999.

[3] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *KDD*, pp. 538–543, 2002.

[4] H. Tong, C. Faloutsos, and J. Pan, "Fast random walk with restart and its applications," in *ICDM*, pp. 613–622, 2006.

[5] C. Wang, F. Jing, L. Zhang, and H. Zhang, "Image annotation refinement using random walk with restarts," in *ACM Multimedia*, pp. 647–650, 2006.

[6] H. Tong, C. Faloutsos, and J. Pan, "Random walk with restart: Fast solutions and applications," *Knowl. Inf. Syst.*, vol. 14, no. 3, pp. 327–346, 2008.

[7] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized PageRank," *PVLDB*, vol. 4, no. 3, pp. 173–184, 2010.
[8] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu, "Fast computation of SimRank for static and dynamic information networks," in *EDBT*, 2010.
[9] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *PVLDB*, vol. 2, no. 1, pp. 718–729, 2009.
[10] P. Sarkar, A. W. Moore, and A. Prakash, "Fast incremental proximity search in large graphs," in *ICML*, pp. 896–903, 2008.
[11] H. Cheng, Y. Zhou, X. Huang, and J. X. Yu, "Clustering large attributed information networks: An efficient incremental computing approach," *Data Min. Knowl. Discov.*, vol. 25, no. 3, pp. 450–477, 2012.