dmo

Dynamic PPR 推导

1 Problem Definition

The problem is to process PPR queries on a dynamic graph with a mixed workload of different types of PPR queries.

Let G = (V,E) be a directed graph. Given a source node s∈V and a decay factor α, a random walk is a traversal of G starting from s, such that at each step of the traversal, it terminates with α probability and, with the other $1-\alpha$ probability, moves to a randomly selected out-neighbor of the current node. For any node t, the Personalized Pagerank (PPR) of t with respect to s, denoted as $\pi(s,t)$, is defined as the probability that a random walk from s stops at t.

The objective of our work is to answer different types of PPR queries while the graph G keeps updating. We assume that operations come from an application and operation arrivals are stochastic. The operations in our work is defined below.

- *Update.* The input of the update operation is an edge (u,v). Let $G_0 = (V_0, E_0)$ be the initial graph. For when processing the i-th update operation, if (u,v) has already existed in the graph G_{i-1}, then remove (u,v) from G_{i-1}, otherwise add (u,v) to G_{i-1}. We let G_i be the graph after process the i-th update operation.

- *Single Source PPR Query.* It returns $\pi(s,t)$ for a give source node s and every node t in G.

- *Top-k PPR Query.* It returns k nodes that have the top-k PPR values with respect to a given node s.

- *One-hop PPR Query.* It returns $\pi(s,t)$ for a given s and every node t that is a one-hop neighbor of s.

The single-source PPR query and the one-hop PPR query returns approximate SSPPR value as follow.

Definition 1.1. (Approximate Whole-Graph SSPPR) Given a source node s, a threshold δ, an error bound ϵ, and a failure probability p_f, an approximate whole-graph SSPPR query returns and estimated PPR $\hat{\pi}(s,v)$ for each node v∈V, such that for any $\pi(s,v) > \delta$,

$$|\pi(s,v) - \hat{\pi}(s,v)| \leq \epsilon \cdot \pi(s,v)$$

holds with at least $1 - p_f$ probability.

Definition 1.2. (Approximate Whole-Graph SSPPR) Given a source node s, a threshold δ, an error bound ϵ, and a failure probability p_f, an approximate one-hop SSPPR query returns and estimated PPR $\hat{\pi}(s,v)$ while v is out neighbor of s, such that for any $\pi(s,v) > \delta$,

$$|\pi(s,v) - \hat{\pi}(s,v)| \leq \epsilon \cdot \pi(s,v)$$

holds with at least $1 - p_f$ probability.

The top-k query returns approximate top-k PPR value as follow.

Definition 1.3. (Approximate Whole-Graph SSPPR) Given a source node s, a threshold δ, an error bound ϵ, and a failure probability p_f, an approximate top-k SSPPR query returns a sequence of k nodes, $v_1, v_2, ..., v_k$, such that with probability $1 - p_f$, for any $i \in [1,k]$ with $\pi(s,v_i^*) > \delta$

$$|\pi(s,v_i) - \hat{\pi}(s,v_i)| \leq \epsilon \cdot \pi(s,v_i)$$

$$\pi(s,v_i) \geq (1-\epsilon) \cdot \pi(s,v_i^*)$$

holds with at least $1 - p_f$ probability, where v_i^* is the node whose actual PPR with respect to s is the i-th largest.

Our work can effectively process the workload composed of query and update operations we defined.

2 Our Solution

2.1 Dynamic Whole-Graph SSPPR

Our method combines forward push and random walk sampling, it first process forward push to reduce the number of required random walk, then perform random walks from nodes which hold none-zero residue. It also maintain a index structure for high-efficiency random walk performing. Our method pre-compute random walks for each node in G, and dynamically update the random walk index while processing update to the graph G.

2.1.1 Optimal Forward Push.

Our work achieve optimal trade-off between forward push and random walk sampling by perform a push step on a node u whenever

$$r(s,u) > \frac{\beta \cdot (d_{out}(u)+1)}{\alpha \cdot K}$$

where K is a constant which is the scale of residue and the number of random walks to perform. As r_{sum} is the sum of the residue of all the nodes in G, and ω is the total number of random walks to perform, we have $K = \frac{\omega}{r_{sum}}$. Algorithm 1 shows the pseudo-code of Optimal Forward Push.

Theorem 2.1. In the forward push phase, let C_p denote the cost of forward pushes that have already been processed, let C_r be the expected cost of random walks sampling phase,

Algorithm 1 Optimal Forward Push

Input: Graph G, source node s, probability α, constant β, K
Output: $\pi^{\circ}(s,u), r(s,u)$ for all $u \in V$

$r(s,s) \leftarrow 1; r(s,u) \leftarrow 0$ for all $u \neq s$;
$\pi^{\circ}(s,u) \leftarrow 0$ for all u;
while $\exists u \in V$ such that $r(s,u) > \frac{\beta \cdot (d_{out}(u)+1)}{\alpha \cdot K}$ **do**
    for *each* $t \in N_{out}(u)$ **do**
        $r(s,t) \leftarrow r(s,t) + (1-\alpha) \cdot \frac{r(s,u)}{|N_{out}(u)|}$
    end for
    $\pi^{\circ}(s,u) \leftarrow \pi^{\circ}(s,u) + \alpha \cdot r(s,u)$;
    $r(s,u) \leftarrow 0$;
end while

let c_p be the cost of change a node's residue, let c_r be the expected cost of performing a single random walk.

Then if $\beta = \frac{c_p}{c_r}$, then the total query cost $Cost = C_p + C_r$ will definitely decrease each time we perform a push on a node u satisfy

$$r(s,u) > \frac{\beta \cdot (d_{out}(u)+1)}{\alpha \cdot K}$$

and vice versa.

Proof. After perform a push on u, C_p is increased by $(d_{out}(u)+1) \cdot c_p$, since push on u changes the residue of all the our neighbor of u and the residue of u. After the push the total residue on the graph is reduced by $\alpha \cdot r(s,u)$, so C_r is reduced by $\alpha \cdot r(s,u) \cdot K \cdot c_r$. The difference of total cost

$$\Delta Cost = (d_{out}(u)+1) \cdot c_p - \alpha \cdot r(s,u) \cdot K \cdot c_r$$

As node u satisfy $r(s,u) > \frac{\beta \cdot (d_{out}(u)+1)}{\alpha \cdot K}$, we can conduct that

$$\Delta Cost < (d_{out}(u)+1) \cdot c_p - \alpha \cdot \frac{\beta \cdot (d_{out}(u)+1)}{\alpha \cdot K} \cdot K \cdot c_r = 0$$

If u satisfy $r(s,u) < \frac{\beta \cdot (d_{out}(u)+1)}{\alpha \cdot K}$, similarly we can conduct that after perform push step the cost difference $\Delta Cost > 0$.

□

We proved that each push step in our method decrease the overall query cost, and further push will increase the cost. In other words, our method achieves optimal trade-off between the cost of forward push and random walk sampling.

Our method obtain c_p and c_r by proceed plenty change a random node's residue and single random walk start from random node. Then we let $\frac{c_p}{c_r}$.

Theorem 2.2.

Proof. During forward push phase, in every push step, we have

$$r(s,u) > \frac{\beta \cdot (d_{out}(u)+1)(u)}{\alpha \cdot K} > \frac{\beta}{\alpha \cdot K}$$

then, in each push step, r_{sum} is reduced by $\alpha \cdot r(s,u)$, which is at least $\frac{\beta}{K}$. So the running time of Algorithm 1 is at most $O(\frac{K}{\beta})$

□

2.1.2 Index Scheme

For a SSPPR query with source node s, our method perform $r(s,u) \cdot K$ from any node u in random walk sampling phase, where r(s,u) denotes u's residue after the forward push phase ends. Meanwhile, our push condition ensures that $r(s,u) \leq \beta \cdot d(u)/(\alpha \cdot K)$. As a consequence, $r(s,u) \cdot K \leq \beta d(u)/\alpha$, we just need to record at most $d(u) \cdot \beta/\alpha$ random walks. So the change of $\epsilon and \delta$ would not effect the length of the random index of every node. Thus while processing queries, the space allocation of the index of every node stays same. If we need to update the value of the index of a single node, we just need to recalculate the end point of each random walk in the updated graph, and do not need to reallocate space for index. To leverage this feature, in our method, we store the whole random index in an array, and store the start and end position of the index of every nodes.

2.1.3 Partial Index Update.

Fora is the state-of-art approach to answer SSPPR queries. However Fora is inefficient on dynamic graphs because every time the graph is updated, fora have to reconstruct the whole random walk index to maintain the correctness of its result.

Our method propose partial index update to efficiently maintain index on dynamic graphs. While a edge (u, v) is insert into or delete from graph G_{i-1}, it will bring inaccuracy to random index of every node in the graph. However different nodes suffer from different inaccuracy. After insertion of (u, v), for any node t in Graph G_i, the closer t and u is, the more inaccurate is the index of t become. In Partial Index Update, out method estimate the inaccuracy of each node in G_i, and select those node which has large index inaccuracy to update. We can prove that, our method can maintain the correctness of SSPPR on dynamic graph with regenerate only a small fraction of random walk index in each update operation.

Definition 2.1. Inaccuracy Rate of Random Walk Index

σ_u is the inaccuracy rate of the random walk index of node u. If we randomly select a random walk from the index, if its destination is v, then $X_i(v) = 1$, otherwise $X_i(v) = 0$. then to $\forall v \in G$

$$|\mathbb{E}[X_i(v)] - \pi(u, v)| \leq \sigma_u$$

During the process of Single Source PPR Query, after the forward push phase is finished, our method fetch ω_i random walks for each node V_i from the index. Where

$$\omega_i = \lceil \frac{r(s, v_i)}{r_{sum}} \cdot \omega \rceil$$

If a random walk end at t, then our algorithm increase $\hat{\pi}(s, t)$ by $\frac{a_i \cdot r_{sum}}{\omega}$, where

$$a_i = \frac{r(s, v_i)}{r_{sum}} \cdot \frac{\omega}{\omega_i}$$

We have

$$r(s,v_i)\cdot(\pi(v_i,t)-\sigma_{v_i})\leq\mathbb{E}[\frac{r_{sum}}{\omega}\sum_{j=1}^{\omega_i}(a_i\cdot X_j(t))]\leq r(s,v_i)\cdot(\pi(v_i,t)+\sigma_{v_i})$$

Theorem 2.3. If $\epsilon'=\theta\epsilon,\ K=\frac{(2\epsilon'/3+2)\cdot log(2/p_f)}{\epsilon'^2\delta},\ \theta$ satisfy that $0\leq\delta\leq 1$, and $\forall u\in G$ we have the index inaccuracy of u satisfy

$$\sigma_u\leq\frac{(1-\theta)\alpha\delta\epsilon K}{m}$$

Then for all $\pi(s,v)>\delta$, for $\hat{\pi}(s,v)$, the SSPPR our method returns, we have

$$|\pi(s,v)-\hat{\pi}(s,v)|\leq\epsilon\cdot\pi(s,v)$$

holds with at least $1-p_f$ probability.

Theorem 2.4. If we process an update with edge (u,v) as input on graph G_{i-1}, then to any vertex s, the increment of its index inaccuracy $\Delta\sigma_s(u,v)$ satisfy

$$\Delta\sigma_s(u,v)\leq\frac{1-\alpha}{\alpha}\cdot\frac{\pi(s,u)}{d_{out}(u)-1}$$

$d_{out}(u)$ is the out-degree of u in G_i.

After an edge (u,v) is inserted to or delete from the graph G_{i-1}, if we want to maintain the correctness of the query result, we must pledge that for all $u\in G_i$, $\sigma_u\leq\Delta\sigma_{max}$. Therefore, our algorithm will perform a backward search to u. By setting $r_{max}=\gamma\cdot\frac{\alpha}{1-\alpha}\cdot(d_{out}(u)-1)\Delta\sigma_{max}$ in backward push, the approximate value of the PPR value from $\forall s\in G$ to u satisfy that

$$\pi(s,u)-r_{max}\leq\widetilde{\pi}(s,u)\leq\pi(s,u)\tag{1}$$

To ensure that for all $u\in G_i$, $\sigma_u\leq\Delta\sigma_{max}$. $\forall s\in G_i$, we regenerate the random walk

index of s if $\widetilde{\pi}(s,u) \geq (1-\gamma) \cdot \frac{\alpha}{1-\alpha} \cdot (d_{out}(u)-1)\Delta\sigma_{max}$.

After insertion, for any node s, if its index is regenerated, then its index inaccuracy $\Delta\sigma_s = 0$, otherwise $\Delta\sigma_s(u,v) \leq \frac{1-\alpha}{\alpha} \cdot \frac{\pi(s,u)}{d_{out}(u)-1}$. If the index of node s is not regenerated, accord to (1), we have

$$\begin{aligned}
\pi(s,u) &\leq \widetilde{\pi}(s,u) + r_{max} \\
&\leq \widetilde{\pi}_{limit} + r_{max} \\
&\leq (1-\gamma) \cdot \frac{\alpha}{1-\alpha} \cdot (d_{out}(u)-1)\Delta\sigma_{max} + \gamma \cdot \frac{\alpha}{1-\alpha} \cdot (d_{out}(u)-1)\Delta\sigma_{max} \\
&\leq \frac{\alpha}{1-\alpha} \cdot (d_{out}(u)-1)\Delta\sigma_{max}
\end{aligned}$$

Therewith we have

$$\begin{aligned}
\Delta\sigma_s(u,v) &\leq \frac{1-\alpha}{\alpha} \cdot \frac{\pi(s,u)}{d_{out}(u)-1} \\
&\leq \frac{1-\alpha}{\alpha} \cdot \frac{\frac{\alpha}{1-\alpha} \cdot (d_{out}(u)-1)\Delta\sigma_{max}}{d_{out}(u)-1} \\
&\leq \Delta\sigma_{max}
\end{aligned}$$

Therefore it is we can conduct that, if initially the index inaccuracy of every nodes in G is zero. With Partial Dynamic Update, after k insertions, for all $u \in G$, $\sigma_u \leq k * \Delta\sigma_{max}$. To maintain the correctness of the query, we just need to set $\Delta\sigma_{max} = \frac{(1-\theta)\alpha\delta\epsilon K}{m \cdot T}$, and regenerate the whole index with every T update operations. Then for any node $u \in G$, σ_u will never exceed $\frac{(1-\theta)\alpha\delta\epsilon K}{m}$, thus accord to Theorem 2.2, the correctness of the query result is guaranteed.

Algorithm 2 shows the pseudo-code of Partial Index Update.

Algorithm 3 shows the pseudo-code of the dynamic whole-graph SSPPR query with Partial Index Update.

Algorithm 2 Partial Index Update

Input: Graph G_i, Update Edge (u, v), probability α

$r(u,t) \leftarrow 1; r(u,t) \leftarrow 0$ for all $t \neq s$;
$\Delta\sigma_{max} = \frac{(1-\theta)\alpha\delta\epsilon K}{m \cdot T}$
$\pi^{\circ}(u,t) \leftarrow 0$ for all t;
$r_{max} \leftarrow \gamma \cdot \frac{\alpha}{1-\alpha} \cdot (d_{out}(u) - 1)\Delta\sigma_{max}$;
while $\exists t \in V_i$ such that $r(u,t) > r_{max}$ **do**
    for *each* $w \in N_{in}(t)$ **do**
        $r(u,w) \leftarrow r(u,w) + (1-\alpha) \cdot \frac{r(u,t)}{|N_{out}(w)|}$
    end for
    $\pi^{\circ}(u,t) \leftarrow \pi^{\circ}(u,t) + \alpha \cdot r(u,t)$;
    $r(u,t) \leftarrow 0$;
end while
for $t \in V$ *with* $\pi^{\circ}(u,t) > (1-\gamma) \cdot \frac{\alpha}{1-\alpha} \cdot (d_{out}(u) - 1)\Delta\sigma_{max}$ **do**
    regenerate the random walk index of t
end for

Algorithm 3 Whole-Graph SSPPR Query with Partial Index Update

Input: Graph G, source node s, probability α

Output: $\pi^{\circ}(s,u), r(s,u)$ for all $u \in V$

Let $\epsilon' = \theta\epsilon$;
Let $K = \frac{(2\epsilon'/3+2) \cdot log(2/p_f)}{\epsilon'^2\delta}$;
Invoke Algorithm 1 with input parameters G, s, α, β, and K; let $\pi^{\circ}(s,v_i)$, $r(s,v_i)$ be the returned reserve and residue of node v_i;
Let $r_{sum} = \sum_{v_i \in V} r(s,v_i)$ and $\omega = r_{sum} * K$;
Let $\hat{\pi}(s.v_i) = \pi^{\circ}(s,v_i)$
for $v_i \in V$ *with* $r(s,v_i).0$ **do**
    Let $\omega_i = \lceil r(s,v_i) \cdot \omega / r_{sum} \rceil$;
    Let $a_i = \frac{r(s,v_i}{r_{sum}} \cdot \frac{\omega}{\omega_i}$;
    for $i = 1$ *to* ω_i **do**
        Perform a random walk W from v_i;
        Let t be the end point of W;
        $\hat{\pi}(s.t) += \frac{a_i \cdot r_{sum}}{\omega}$;
    end for
end for
return $\hat{\pi}(s.v_1), ..., \hat{\pi}(s.v_n)$

2.1.4 Lazy Index Update

In the random walk sampling phase, if the i-th node in graph G_i v_i has index inaccuracy, then the absolute error that v_i bring to the approximate SSPPR is $r(s,v_i) \cdot \sigma_{v_i}$. We can observe that, in a whole-graph SSPPR query, if a node has large index accuracy, but also

has small residue value, it bring little error to our result. There are a lot of this kind of nodes in a query. With partial index update,the index of this kind of node is regenerated, which hardly effect the absolute error query result and is actually unnecessary. 可以加入实验，*top a*%的node带来了99%的不准确度.

We can further improve the performance of our method with lazy index update. In lazy index update, we maintain a vector I to store the current upper bound of the index inaccuracy of all nodes in G_i. Therefore we have $I_i(u) \geq \sigma_u$. Initially $I_0 = \mathbf{0}$. In update operation, we perform a backward search to estimate the upper bound of index inaccuracy increment $\Delta\sigma_u$ of $\forall u \in G_{i-1}$. Then after update operation we have $\forall u \in Gi : I_i(u) \leftarrow I_{i-1}(u) + \Delta\sigma_u$. In lazy index update, we do not regenerate any index in update operation, but regenerate index only when it is necessary.

Let $e(v)$ denote the absolute error node u bring to the result of whole-graph SSPPR query due to its index accuracy. Let $\hat{e}(v) = I(v) \cdot r(s, v)$, then we have $e(v) \leq \hat{e}(v)$. Let e_{sum} denote the sum of absolute error due to index accuracy, then $e_{sum} = \sum_{v_i \in G} e(v_i) \leq \sum_{v_i \in G_i} I_i(v_i) \cdot r(s, v_i)$. Each time we process a whole-graph SSPPR query, we have

$$|\pi(s, v_i) - \hat{\pi}(s, v_i)| \leq \epsilon \cdot \pi(s, v_i) + e_{sum}$$

holds with at least $1 - p_f$ probability if $\pi(s, v_i) > \delta$.

If the given $\epsilon = \epsilon_0$, to ensure $|\pi(s, v_i) - \hat{\pi}(s, v_i)| \leq \epsilon_0 \cdot \pi(s, v_i)$ holds with at least $1 - p_f$ probability if $\pi(s, v_i) > \delta$. We just need to process the query with $\epsilon = \theta \cdot \epsilon_0$, where $\theta \in (0, 1)$. Then we have

$$|\pi(s, v_i) - \hat{\pi}(s, v_i)| \leq \theta \cdot \epsilon_0 \cdot \pi(s, v_i) + e_{sum}$$

holds with at least $1-p_f$ probability if $\pi(s, v_i) > \delta$. We can observe that, if we can guarantee that $e_{sum} < (1 - \theta) \cdot \epsilon_0\delta$, then we will have

$$|\pi(s, v_i)-\hat{\pi}(s, v_i)| \leq \theta\cdot\epsilon_0\cdot\pi(s, v_i)+(1-\theta)\cdot\epsilon_0\cdot\delta \leq \theta\cdot\epsilon_0\cdot\pi(s, v_i)+(1-\theta)\cdot\epsilon_0\cdot\pi(s, v_i) = \epsilon_0\cdot\pi(s, v_i)$$

holds with at least $1 - p_f$ probability. Then our query result will be correct.

To make sure that $e_{sum} < (1-\theta)\cdot\epsilon_0\cdot\delta$, after the forward push phase is over, we repeatedly select the node v_i with largest $\hat{e}(v_i)$ in G_i, and regenerate the random walk index of v_i to let $\hat{e}(v_i) \leftarrow 0$, until $e_{sum} < (1-\theta)\cdot\epsilon_0\cdot\delta$.

Algorithm 4 shows the pseudo-code of Lazy Index Update.

Algorithm 5 shows the pseudo-code of the dynamic whole-graph SSPPR query with Lazy Index Update.

Algorithm 4 Lazy Index Update

Input: Graph After Update G, update edge (u, v), probability α, index inaccuracy I

$r(u,t) \leftarrow 1; r(u,t) \leftarrow 0$ for all $t \neq s$;
$\Delta\sigma_{max} = \frac{(1-\theta)\alpha\delta\omega\epsilon}{m}$
$\pi^\circ(u,t) \leftarrow 0$ for all t;
$r_{max} \leftarrow \gamma \cdot \frac{\alpha}{1-\alpha} \cdot (d_{out}(u) - 1)\Delta\sigma_{max}$;
while $\exists t \in V$ such that $r(u,t) > r_{max}$ **do**
    for *each* $w \in N_{in}(t)$ **do**
        $r(u,w) \leftarrow r(u,w) + (1-\alpha) \cdot \frac{r(u,t)}{|N_{out}(w)|}$
    end for
    $\pi^\circ(u,t) \leftarrow \pi^\circ(u,t) + \alpha \cdot r(u,t)$;
    $r(u,t) \leftarrow 0$;
end while
for $t \in V$ **do**
    Let $\Delta\sigma_t = r_{max} + \pi^\circ(u,t)$;
    $I(t) \leftarrow I(t) + \Delta\sigma_t$;
end for

Algorithm 5 Whole-Graph SSPPR Query with Lazy Index Update

Input: Graph G, source node s, probability α, index inaccuracy I

Output: $\pi^{\circ}(s,u), r(s,u)$ for all $u \in V$

Let $\epsilon' = \theta\epsilon$;

Let $K = \frac{(2\epsilon'/3+2)\cdot log(2/p_f)}{\epsilon'^2\delta}$;

Invoke Algorithm 1 with input parameters G, s, α, β, and K; let $\pi^{\circ}(s,v_i)$, $r(s,v_i)$ be the returned reserve and residue of node v_i;

Let $r_{sum} = \sum_{v_i \in V} r(s,v_i)$ and $\omega = r_{sum} * K$;

Let $\hat{e}(v_i) = r(s,v_i) \cdot I(v_i)$ and $\hat{e}_{sum} = \sum_{v_i \in V} \hat{e}(v_i)$

while $\hat{e}_{sum} > (1-\theta) \cdot \epsilon_0\delta$ **do**

    $\hat{e}(v_k) = max\{\hat{e}v_1, ..., \hat{e}v_n\}$;

    regenerate the random walk index of node v_k;

    $\hat{e}_{sum} \leftarrow \hat{e}_{sum} - \hat{e}(v_k)$, $\hat{e}(v_k) \leftarrow 0$;

end while

Let $\hat{\pi}(s.v_i) = \pi^{\circ}(s,v_i)$

for $v_i \in V$ *with* $r(s,v_i).0$ **do**

    Let $\omega_i = \lceil r(s,v_i) \cdot \omega / r_{sum} \rceil$;

    Let $a_i = \frac{r(s,v_i}{r_{sum}} \cdot \frac{\omega}{\omega_i}$;

    for $i = 1$ *to* ω_i **do**

        Perform a random walk W from v_i;

        Let t be the end point of W;

        $\hat{\pi}(s.t) += \frac{a_i \cdot r_{sum}}{\omega}$;

    end for

end for

return $\hat{\pi}(s.v_1), ..., \hat{\pi}(s.v_n)$

2.2 Amortized Time Complexity of Lazy Update

In this section, we will analyze the amortized time complexity of lazy update.

For each update operation, an edge (u, v) is inserted in or deleted from the graph. According to Theorem 2.4, for any vertex s, the increment of its index inaccuracy $\delta_s(u, v)$ satisfy $\Delta\sigma_s(u, v) \leq \frac{1-\alpha}{\alpha} \cdot \frac{\pi(s,u)}{d_{out}(u)-1}$. If we perform a backward push with $r_max = \epsilon_b$, then for every node s we can get approximation of the ppr from s to u satisfy $\pi(s, u) - \epsilon < \widetilde{\pi}(s, u) < \pi(s, u)$, then we have $\pi(s, u) < \widetilde{\pi}(s, u) + \epsilon$, so

$$\delta_s(u, v) < \hat{\delta}_s(u, v) = \frac{1-\alpha}{\alpha} \cdot \frac{\widetilde{\pi}(s, u) + \epsilon}{d_{out}(u) - 1}$$

For the i-th update, we let $I_i = I_{i-1} + \hat{\delta}_s(u, v)$, so that for any node $s \in G_i$, $I_i(s)$ would always be the upper bound of the index inaccuracy of s.

In an update, for a random node s, and a random edge (u, v), the expected increment for I(s) is

$$\mathbb{E}[\hat{\delta}_s(u, v)] = \mathbb{E}[\frac{1-\alpha}{\alpha} \cdot \frac{\widetilde{\pi}(s, u) + \epsilon}{d_{out}(u) - 1}] \leq \frac{1-\alpha}{\alpha} \cdot (\mathbb{E}[\widetilde{\pi}(s, u)] + \epsilon)$$

For a random node s, $\mathbb{E}[\pi(s, u)] = \frac{\sum_{s_i \in G} \pi(s_i, u)}{n} = \frac{pr(u)}{n}$. For a random node s, and a random edge (u, v),

$$\mathbb{E}[\pi(s, u)] = \mathbb{E}[\frac{pr(u)}{n}] = \frac{\frac{\sum_{u_i \in G} pr(u)}{n}}{n} = \frac{1}{n}$$

As $\widetilde{\pi}(s, u) < \pi(s, u)$, we can derive that

$$\mathbb{E}[\hat{\delta}_s(u, v)] \leq \frac{1-\alpha}{\alpha} \cdot (\frac{1}{n} + \epsilon)$$

Then in each update, we let the amortized increment of I be $\frac{1-\alpha}{\alpha} \cdot (\frac{1}{n} + \epsilon)$. In a SSPPR query with lazy update, assume current graph is G_h, for a random node t

$$
\begin{aligned}
&\mathbb{E}[e(t)] \\
&\leq \mathbb{E}[I_h(t) \cdot r(t)] \\
&\leq \frac{1-\alpha}{\alpha} \cdot (\frac{1}{n} + \epsilon) \cdot N_t \cdot \frac{r_{sum}}{n} \\
&\leq \frac{1-\alpha}{\alpha} \cdot (\frac{1}{n} + \epsilon) \cdot N_t \cdot \frac{m}{\alpha n K}
\end{aligned}
$$

N_t is the update operation processed after the last time the index of t is regenerated.

Then

$$\mathbb{E}[e_{sum}] \leq \frac{m(1-\alpha)(1+n\epsilon_b)}{\alpha^2 nK} \cdot \frac{\sum_{t \in G_h} N_t}{n}$$

While h is large enough, we assume on average the index of each node is regenerated with every X times of update operation, then amortizely $\sum_{t \in G_h} N_t = \frac{nX}{2}$. Then we have

$$\mathbb{E}[e_{sum}] \leq \frac{m(1-\alpha)(1+n\epsilon_b)}{\alpha^2 nK} \cdot \frac{X}{2}$$

In lazy update we ensure that $e_{sum} < (1-\theta)\epsilon_0 \delta$, then amortizely

$$\frac{m(1-\alpha)(1+n\epsilon_b)}{\alpha^2 nK} \cdot \frac{X}{2} \leq (1-\theta)\epsilon_0 \delta$$

we can derive that

$$X \leq \frac{2\alpha^2 K(1-\theta)\epsilon_0}{m(1-\alpha)(1+n\epsilon_b)}$$

So amortizely with X updates, the index of every node on the graph is regenerated once. Then, the amortized time complexity of index regenerating for a single update operation is

$$O(\frac{n}{X}) = \frac{nm(1-\alpha)(1+n\epsilon_b)}{2\alpha^2 K(1-\theta)\epsilon_0}$$

2.3 Dynamic Top-k PPR

The Dynamic Top-k PPR of our method is based on the top-k PPR in FORA. Algorithm 6 shows the pseudo-code of Dynamic Top-k PPR.

Algorithm 6 Dynamic Top-k PPR

Input: Graph G, source node s, probability α

Output: k nodes with the highest PPR scores

1: **for** $\delta = \frac{1}{k}, \frac{1}{2k}, ..., \frac{1}{n}$ **do**
2: Invoke Dynamic Whole Graph PPR with G,s,α and set fail probability $p'_f = \frac{p_f}{n \cdot logn}$
3: Let $C = v'_1, ..., v'_k$ be the set that contains the k nodes with the top-k largest lower bounds;
4: Let $LB(u)$ and $UB(u)$ be the lower and upper bounds of $\pi(s, u)$;
5: **if** $UB(v'_i) < (1+\epsilon) \cdot LB(v'_i)$ for $i \in [1, k]$ **and** $LB(v'_k) \geq \delta$ **then**
6: Let U be the set of nodes $u \in V \setminus C$ such that $UB(u) > (1+\epsilon) \cdot LB(v'_k)$;
7: **if** $\nexists u \in U$ **such that** $UB(u) < (1+\epsilon) \cdot LB(u)/(1-\epsilon)$ **then**
8: return $v'_1, v'_2, ..., v'_k$ and their estimated PPR;
9: **end if**
10: **end if**
11: **end for**

δ is initially set as a large value. In each iteration we invoke dynamic whole graph PPR with a smaller δ than last iteration, until δ is smaller enough to calculate the correct approximate result of the k-th largest PPR. Let $\delta_{min} = \frac{1}{n}$, then in each iteration after dynamic whole graph PPR with partial index update or lazy index update we have

$$Pr[|\pi(s,t) - \hat{\pi}(s,t)| \leq \theta \cdot \epsilon \cdot \pi(s,t) + (1-\theta) \cdot \epsilon \cdot \delta_{min}]$$

holds with at least $1 - p'_f$ probability. Because $\delta_{min} < \delta < \pi(s,t)$, we have

$$Pr[|\pi(s,t) - \hat{\pi}(s,t)| \leq \epsilon \cdot \pi(s,t)]$$

holds with at least $1 - p'_f$ probability.

The difference between dynamic Top-k PPR and regular Top-k PPR is that while updating the bounds we have ω_j is the *omega* calcuated by our method (Algorithm 4 Line 2 or Alorithm). Let $\epsilon' = \theta\epsilon$. We have

$$Pr[|\pi(s,t) - \hat{\pi}(s,t)| \geq \epsilon \cdot \pi(s,t)] \leq 2 \cdot \exp(-\frac{\epsilon'^2 \cdot \omega_j \cdot \pi(s,t)}{2 + 2a \cdot \epsilon'/3})$$

Let p'_f equal the RHS of above inequality. We have $\epsilon' \geq \sqrt{\frac{3log(2/p'_f)}{\omega_j \cdot max\pi^{\circ}_j(s,v), LB(j-1)(s,v)}}$

By setting $\epsilon_j = \frac{1}{\theta} \cdot \sqrt{\frac{3log(2/p'_f)}{\omega_j \cdot max\{\pi^{\circ}_j(s,v), LB_{j-1}(s,v)\}}}$, we can derive that $\hat{\pi_j}(s,v)/(1+\epsilon_j) \leq \pi(s,v) \leq \hat{\pi_j}(s,v)/(1-\epsilon_j)$. And by Fora we also have

$$\lambda_j = \frac{2/3 \cdot log(2/p'_f)}{2\omega_j} + \frac{\sqrt{\frac{4}{9} r^2_{sum} \cdot log^2(2/p'_f) + 8 r_{sum} \cdot \omega_j \cdot log(2/p'_f) \cdot UB_{j-1}(v)}}{2\omega_j}$$

Then in each iteration, the following two inequalities hold simultaneously:

$$\hat{\pi_j}(s,v)/(1+\epsilon_j) \leq \pi(s,v) \leq \hat{\pi_j}(s,v)/(1-\epsilon_j)$$
$$\hat{\pi_j}(s,v) - \lambda_i \leq \pi(s,v) \leq \hat{\pi_j}(s,v) + \lambda_i$$

We leverage the upper equations to update the upper bounds and lower bounds of nodes in each iteration.

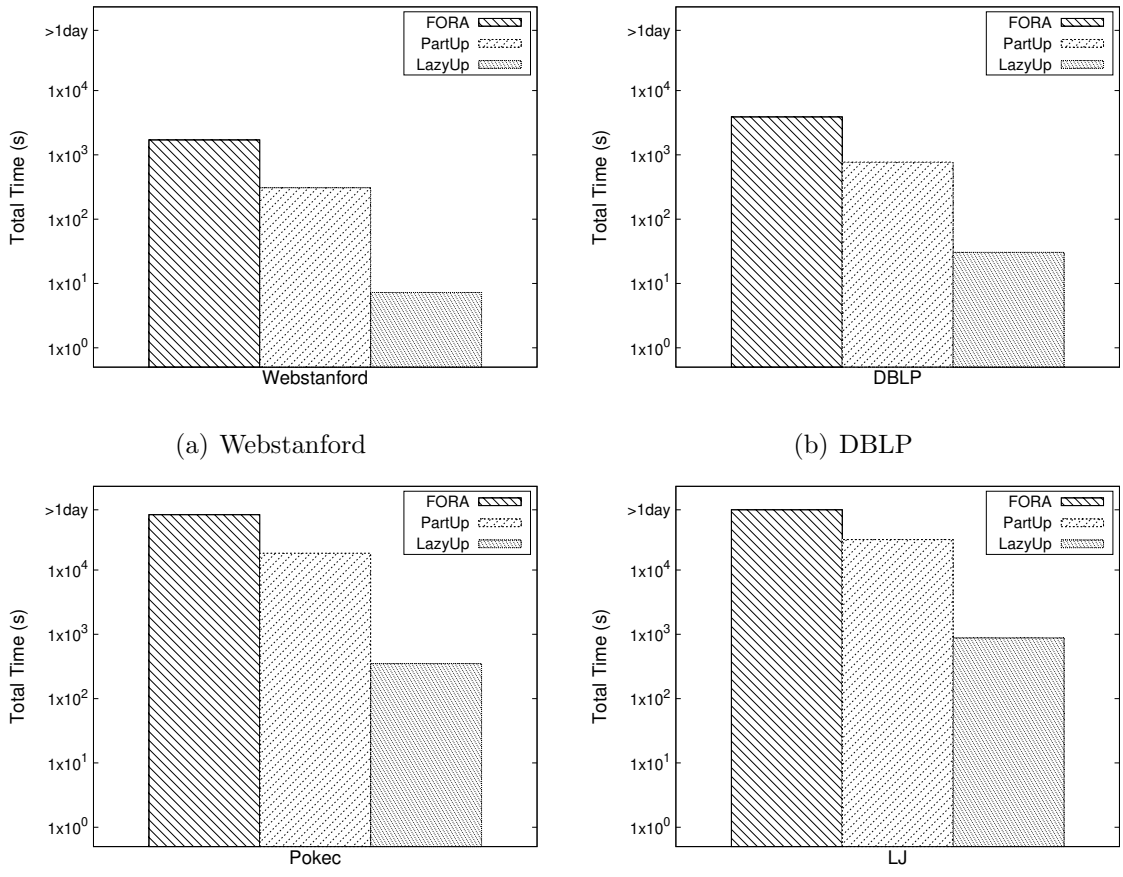
One of the optimization is that, in FORA, the initial value of δ is $\frac{1}{2}$, which however is to large. In our method, the initial value of δ is set as $\frac{k}{1}$. The intuition is that, $L(v'_k) > \frac{1}{k}$, and the iteration of δ stop only when $LB(v'_k) \geq \delta$, so the iteration with $\delta > \frac{k}{1}$ is meaningless.

2.4 One-hop PPR

The procession of one-hop PPR query is same with whole graph PPR query. Except that δ is set as $\alpha(1-\alpha)/d_{out}(s)$

3 Evaluation

We evaluate the effectiveness of our method by process a workload with 200 updates and 200 SSPPR queries. The result is shown below.



(a) Webstanford

(b) DBLP

(c) Pokec

(d) LJ

Figure 1: Effectiveness on Dynamic Graph