Towards PageRank Update in a Streaming Graph by Incremental Random Walk

ZHIPENG SUN1,2, GUOSUN ZENG1,2, (Senior Member, IEEE), AND CHUNLING DING3

1Embedded System and Service Computing Key Laboratory of Ministry of Education, Tongji University, Shanghai 200070, China
2Department of Computer Science and Technology, Tongji University, Shanghai 200070, China
3College of Chemical Science and Engineering, Tongji University, Shanghai 200070, China

Corresponding author: Guosun Zeng (gszeng@tongji.edu.cn)

This work was supported in part by the Subproject of National Seafloor Observatory System of China under Grant 2970000001/001/016.

ABSTRACT As the internet and the Internet of Things (IoT) have been widely applied in many application fields, a large number of graphs are continuously produced and change over time, which leads to difficulties in graph analysis and utilization. This paper studies a PageRank update algorithm for a streaming graph using incremental random walk. We focus on the information about the local changes of nodes and edges in the current graph, analyze the impact of such local changes on this current graph, and then use the idea behind wave propagation theory to seek and determine all affected nodes that need to be updated their PageRank in the new graph. For new nodes, the existing nodes in the current graph that are connected with these new nodes are aggregated into a supernode, and the PageRank of the new nodes is solved in the new graph with the supernode. Finally, we conduct a series of experiments on real-world and synthetic graph datasets. Compared with the state-of-the-art incremental computing algorithm, our algorithm not only ensures the accuracy of calculating the PageRank in a large streaming graph, but also speeds up the computational process by avoiding many redundant computations.

INDEX TERMS Streaming graph, PageRank of nodes, wave propagation, incremental computation, random walk.

I. INTRODUCTION

To evaluate the importance of web pages, the concept of PageRank in Google was first introduced by Page and Brin [1]. Because web pages and their hyperlinks can be treated as nodes and edges in a directed graph respectively, PageRank is also used in other graphs and in many application fields. For example, in e-commerce, a PageRank of products can help to recommend the relevant products to clients efficiently. In intelligent transportation systems, detecting the PageRank in an urban traffic network can be used to optimize the travel path. In social networks, people can find a user’s friends by using PageRank, and if this user is a criminal, the policemen can easily determine the criminal’s associates. In biological networks, geneticists can find some common oncogenes by using PageRank, which can avoid congenital diseases effectively. Generally speaking, the above graphs are very large and dynamically change over time [2]. Moreover, the fixed PageRank in dynamic streaming graphs cannot always be always valid because their structures change, especially when the addition and deletion of nodes or edges occurs. In this situation, PageRank needs to be updated after dynamic changes [3]. However, it is mostly a small-scale change at every moment of attention compared with the whole streaming graph, e.g., the total number of articles added was less than 4% for English Wikipedia in 2019 [4]. If we compute the PageRank of all nodes in a graph from scratch every time, this process will continuously produce rather redundant computation continuously. Computing PageRank in this way is not only inefficient but also wastes considerable computing resources. Additionally, even if the parallelism of this method could be realized in a parallel distributed environment, the execution time would not be advisable for real-time applications. However, PageRank in dynamic streaming graphs needs to be updated in real-time, which is a nontrivial challenge. Therefore, the incremental calculation method of a PageRank has emerged in some current studies [5, 6]. They make good use of the previous results and compute the PageRank only, which needs to be updated incrementally every time [6]. However, to the best of our knowledge, these existing incremental methods still have some shortcomings in identifying the nodes that
need to update their PageRank, which makes it difficult to regulate a trade-off between improving computing efficiency and minimizing the calculation error. In this paper, we use the idea of a random walk and design a novel incremental computation algorithm for updating PageRank in a dynamic streaming graph. Our main contributions can be summarized as follows:

(1) Local changes of nodes and edges in a streaming graph are particularly concerning, which may cause the PageRank of some nodes to change. Thus, we apply wave propagation theory to discover the affected local area in such graphs at the present moment and develop a novel algorithm to determine all affected nodes that can reduce error and the amount of calculation at the same time.

(2) The PageRank of the affected nodes is calculated in an incremental manner, which is based on the Monte Carlo idea. We leverage the information about random walks and about changes including adding and deleting nodes or edges in order to identify the number of the changed random walk paths and calculate the probabilities of visiting these nodes. Then, an optimized PageRank update method for the affected nodes is presented to avoid redundant computation and accelerate the PageRank update.

(3) For newly added nodes, the existing nodes in the current graph that are connected with these new nodes are aggregated into a supernode. These new nodes with the supernode form a smaller new graph where the PageRank of newly added nodes is calculated at a very low computational cost.

The rest of this paper is structured as follows. Section II investigates the related work. Section III presents the problem description about updating PageRank in a large dynamic streaming graph. Section IV proposes a novel approach to finding all affected nodes and conducts a deep analysis. Section V elaborates on an incremental computation of PageRank in a streaming graph. Some experiments are conducted to show the advantage of our approach over traditional techniques in Section VI. Section VII concludes this paper.

II. RELATED WORK

When we review PageRank computation, the traditional methods can be divided into two categories: power iteration methods [7] and Monte Carlo methods [8], [9]. The former converges to the PageRank by iterative operation of the graph connection matrix, and the latter uses a random walk to approximate the PageRank. Both methods can be used to calculate PageRank for static and dynamic graphs, and many extension methods based on them have been developed.

For static graphs, Desikan et al. [10] used a power iteration method to calculate all nodes’ PageRank for a static graph. Kamvar et al. [11] used the quadratic extrapolation method to remove the nonmain eigenvectors in the current iteration and accelerate the convergence process of iterative calculation. Arnal [12] presented a parallel vision of the power iteration method to reduce the execution time for PageRank computation. However, according to the Monte Carlo idea, Nigam [13] designed a method to approximate PageRank based on the Markov model, and verified its effectiveness. Fushimi et al. [14] proposed a p-avg approach to obtain the approximate PageRank scores, which improved the speed of PageRank computation.

Considering a dynamic graph, the easiest way is that the methods of calculating PageRank for a static graph can be used to calculate PageRank for a dynamic graph. Christian et al. [15] turned a dynamic graph into a series of snapshots, treated these snapshots as static graphs, and calculated nodes’ PageRank related to these static graphs by the Gauß-Seidel method. Gonzalez [16] regarded a dynamic graph in batches over time as a static graph and computed nodes’ PageRank. The sliding time window was introduced by Bahmani et al. [17] to deal with the dynamic streaming graph for computing PageRank. Essentially, these methods convert dynamic graphs into many static graphs, and then calculate all nodes’ PageRank again and again. If the time window is very short and too many static graphs are generated, these methods are time-consuming.

Generally, in some application fields, a dynamic graph often locally changes on a small scale, and a few nodes or edges are added into or removed away from the current graph during a period of attention. For this reason, only a small number of nodes’ PageRank will be affected. To speed up the solution, one only needs to recalculate the PageRank of this part of the affected nodes rather than all of the graph nodes. Inspired by this idea, some incremental computing frameworks and platforms have emerged, such as GraphIn [18], Tornado [19] and DZIG [20]. Moreover, many incremental update PageRank algorithms have been proposed. Kim [21] updated the PageRank of the affected nodes by using an incremental iteration method. Chien et al. [22] proposed a novel method for discovering the affected nodes, showing that the influence gradually decreases as it passes through its reachable nodes. Zhang [23] presented an incremental power iteration method, I-PageRank, which avoided computing PageRank from scratch. MaSherry [24] invented a new power iteration method that can accelerate the convergence of updated iterations. Bahmani et al. [25] proposed a proportional probing method, which could determine the existing nodes with high PageRank changes and take a small affected portion of the graph to update the PageRank of these nodes. Yu et al. [26] designed an incremental random walk algorithm IRWR, which mainly focused on the edges that were constantly changed and found top-k nodes that were affected by these edges to update PageRank. Prasanna et al. [27] treated a strongly connected components as an affected area in a graph and calculated the PageRank of the local affected nodes at low computational cost. Atish et al. [8] modified the PageRank of all nodes by adjusting the affected random walk paths in a streaming manner. Liao et al. [28] addressed
an incremental PageRank algorithm, IMCPR, based on the Monte Carlo method, which obtained lower computational complexity $O((m + n/c)/c)$, where $m$, $n$ and $c$ are the number of changed nodes, the number of changed edges and the reset probability, respectively. In addition, currently distributed computing systems are widely used to speed up big graph processing. Mariappan et al. [29] proposed an incremental BSP processing model to calculate the PageRank in a big streaming graph.

Although the above methods for a big dynamic graph can save computational time, their disadvantages include possibly being inaccurate or unreasonable when finding the affected node area. Riedy [3] proposed a PageRank updating method based on iterative refinement. When a dynamic graph changed, the error of PageRank was compensated in the process of updating PageRank to ensure that the updating result was accurate. Zhang et al. [30] presented two dynamic versions of forward push and reverse push based on the power iteration method, which are more accurate than those in Reference [17]. Yoon et al. [31] proposed a fast and accurate OSP algorithm, which updated PageRank on dynamic graphs by random walk with a restart and set the restart probability to balance accuracy and calculation time. Reference [28] assumed that the random walk path did not visit the same node repeatedly, which was inconsistent with the fact. Therefore, the calculated PageRank had a large error, and the error increased continuously with the dynamic change of the graph. Zhan et al. [9] proposed the revisit probability model to truly reflect the random walk process. When a dynamic graph changed, the PageRank was updated based on the original random walk information.

The above works all attempted to accelerate the calculation process of PageRank while ensuring high accuracy, but there was still no a good trade-off between accuracy and efficiency. Therefore, this work studies an algorithm for PageRank update in a streaming graph using incremental random walk, and realizing the parallelism of this algorithm. Our goal is to achieve high precision and high efficiency at the same time.

### III. CONCEPTS AND PROBLEM DESCRIPTION

#### A. TRADITIONAL METHOD TO CALCULATE PAGERANK

In this paper, we let $G = (V, E)$ be a directed graph, where $V$ and $E$ are the sets of nodes and edges, respectively. $|V|$ is the number of nodes and $|E|$ is the number of edges. For two nodes $u, v \in V$, if there exists a directed edge $(u, v) \in E$, $Out_u$ is the set of $u$'s outgoing neighbors, and $v \in Out_u$, we let $A$ be the adjacency matrix related to $G$ as follows:

$$A(u, v) = \begin{cases} 1, & \text{if } (u, v) \in E \\ 0, & \text{otherwise.} \end{cases}$$  

(1)

We let $D$ be the diagonal matrix of nodes' out-degrees. Then, the transition matrix related to $G$ is denoted as $C = A^T D^{-1}$. When $X^{(0)}$ is an initial vector with $|V|$ entries, the value of each entry in $X^{(0)}$ is $1/|V|$. Then the PageRank of each node in $G$ can be $PR = (pr_1, pr_2, \ldots, pr_{|V|})^T$, which can be calculated as follows:

$$E \left\{ X^{(k+1)} = \alpha CX^{(k)} + \frac{(1-\alpha)e^T}{|V|} \right\} \leq \epsilon$$

(2)

$$PR = (pr_1, pr_2, \ldots, pr_{|V|})^T = X^{(k+1)}$$

where $\alpha$ is the probability that any node’s outgoing neighbors are visited, $e$ is a $|V|$-dimensional unit vector, $\epsilon$ is a threshold satisfying the convergence requirement, and $pr_j$ is the calculated PageRank of node $v_j$.

#### B. THE MONTE CARLO IDEA TO CALCULATE PAGERANK

The Monte Carlo idea has been reported to approximate the PageRank of a graph [32]. To clarify the calculation process, we first introduce some concepts.

**Definition 1 (Random Walk, RW):** In a graph $G = (V, E)$, a random walk is a type of random process, during which many steps need to be taken as follows: a walker chooses a node $v \in V$ as a starting node and randomly walks along the out-edge of this node with a probability of $\alpha$ at each step. In the same way, if the walker keeps walking, the total number of steps must not exceed $R$. Therefore, a walker may stop with a probability of $1 - \alpha$ after each step, or terminate where the node has no any out-edge. To approximate the PageRank of each node more effectively, $M$-times random walks starting from each node are performed.

**Definition 2 (Random Walk Path, RWP):** In a graph $G = (V, E)$, a random walk path is a continuous path starting from any node to a reachable node. It can be denoted as $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$, where $v_i \in V, (v_i, v_{i+1}) \in E, i \in [1, k]$.

For each node $v_i$, we let $s_{v_i}$ be the total number of times that all RWPs have visited $v_i$ using the random walk method in Definition 1. Then, we can approximate the PageRank of $v_i$, denoted by $\tilde{pr}_{v_i}$ as follows:

$$\tilde{pr}_{v_i} = s_{v_i}/(|V|M).$$

(3)

Let $w_{v_i}$ be the number of all RWPs passing through node $v_i$. If each RWP visits $v_i$ only once, then $s_{v_i} = w_{v_i}$. However, there may exist a cycle in $G$, which means that an RWP may visit the same node many times. In this case, $s_{v_i} > w_{v_i}$. To discuss the relationship between $s_{v_i}$ and $w_{v_i}$, let’s take an example is shown in Fig.1, where $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$ is a random walk path that starts from $v_1$.

Here, we use the variable $r_{v_1v_2}$ to represent the probability that the path starting from node $v_1$ passes through an out-edge $(v_1, v_2)$ and returns to $v_1$, then $r_{v_1v_2} = \alpha \times \alpha \times 1/2 \times \alpha \times 1/3$. In general, if $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_1 \rightarrow \ldots \rightarrow v_1$ is an RWP, $r_{v_1v_2} = \alpha^{k-1} \prod_{i=1}^{k} 1/|Out_{v_i}|$ where $L$ is the length of this path, and $Out_{v_i}$ is the set of node $v_i$’s outgoing neighbors. Therefore, the probability of RWs that start from node $v_1$ and return to $v_1$ can be calculated as follows:

$$r_{v_1} = \sum_{v_2 \in Out_{v_1}} \frac{1-\alpha}{|Out_{v_1}|} r_{v_1v_2}.$$
Thus, we can obtain the relationship among \( r_{v_i}, w_{v_i}, \) and \( s_{v_i} \) as follows:

\[ s_{v_i} = w_{v_i} \left( 1 + r_{v_i} + r_{v_i}^2 + \ldots \right) = w_{v_i} \frac{1 - (r_{v_i})^n}{1 - r_{v_i}} \]

\[ = \left[ \frac{w_{v_i}}{1 - r_{v_i}} \right]. \quad (5) \]

**Proposition 1:** In a graph \( G = (V, E) \), for each node \( u \in V \), we let the PageRank of \( u \) be \( \hat{pr}_u \) which is calculated by Formula 2. If \( \hat{pr}_u \) is calculated by Formula 3, then \( \hat{pr}_u \) approximates to \( pr_u \).

**Proof:** Assume that the true PageRank of node \( u \) is denoted as \( \pi_u \), and \( pr_u \) is calculated by using the basic method that is equal to \( \pi_u \). A “random walk” that we define is based on the Monte Carlo methods. We walk randomly along the directed edges in the graph, and the probability of passing through any node \( u \) tends to be stable. This probability of node \( u \), denoted as \( \hat{pr}_u \), is treated as the approximate PageRank of \( u \) [33]. Since \( \hat{pr}_u = \pi_u \), then \( \hat{pr}_u \) approximates to \( pr_u \). \( \square \)

### C. PAGERANK UPDATE PROBLEM

**Definition 3 (Streaming Graph, SG):** A streaming graph is a dynamic graph changing over time \( 1, 2, 3, \ldots, t \ldots \), denoted as \( G^1, G^2, G^3, \ldots, G^t, \ldots \), where \( G^2 \) evolves from \( G^1 \), \( G^3 \) evolves from \( G^2 \), \( G^t \) evolves from \( G^{t-1} \), etc.

Formally, \( G^t = G^{t-1} + \Delta G^t \), where \( \Delta G^t \) is a small graph that contains all the change information about node or edge insertion or deletion at time \( t \). \( \Delta G^t = G^t - G^{t-1} \) can be generated and informed by an online graph computing system. More concretely, we allow \( \Delta G^t = (\Delta V^t, \Delta E^t) = (\Delta V^0, \Delta V^+, \Delta V^-, \Delta E^+, \Delta E^-) \), where \( \Delta V^0 \) is the set of nodes with one or more edges added or deleted in \( G^{t-1} \), \( \Delta V^+ \) is the set of all new nodes, \( \Delta V^- \) is the set of all deleted nodes at time \( t \), \( \Delta E^+ \) is the set of all new edges, and \( \Delta E^- \) is the set of all deleted edges at time \( t \).

An example is shown in Fig. 2. A black-circle represents the original node, a green rectangle indicates that a new node has been added, and a green arrow indicates that a new edge has been added. A dotted green circle indicates the deletion of a node and the dotted green arrows indicate the deletion of an edge.

Using traditional methods to calculate PageRank is time-consuming. Moreover, a large number of redundant processing operations occur when calculating PageRank from scratch with the continuous expansion of the data scale in a streaming graph. Therefore, we need to seek some new methods to improve. According to the actual situation, \( \Delta G^t \) is much smaller than \( G^t \), and the PageRank in \( G^t \) may be very similar to that in \( G^{t-1} \). As in Fig. 2(a), a large number of computational operations are wasted on some nodes with almost unchanged PageRank, meaning repetitive and redundant computations. As a result, we will study an incremental method to calculate PageRank as shown in Fig. 2(b), where only a few red nodes need to update their PageRank.

### IV. AFFECTED AREA DUE TO GRAPH CHANGES

If a dynamic graph does not change, the PageRank of all nodes does not need to be recalculated. However, even if a small number of nodes or edges are added or deleted, the PageRank of some related local nodes are affected. In this section, we discuss how to determine the affected area.

#### A. WAVE PROPAGATION PHENOMENON

Throughout the process of sound propagation, the sound intensity gradually decreases as sound travels farther. It has been reported that the relationship between sound intensity and propagation distance is \( P_x = P_0 e^{-\beta x} \) [34], where \( P_0 \) is the sound intensity at the origin, \( P_x \) is the sound intensity at location \( x \) far from the origin, and \( \beta \) is the attenuation coefficient. Generally, \( \beta = 0.0255 \). According to this theory, we can apply it to analyze the dynamic change and local impact in a streaming graph.
B. FINDING AFFECTED NODES FOR UPDATING PAGERANK

As stated in Section III-C, $\Delta G^i = (\Delta V^i, \Delta E^i) = (\Delta V_{0,i} + \Delta V_{+,i} + \Delta V_{-,i}, \Delta E_{+,i} + \Delta E_{-,i})$. If $\Delta V_{0,i} = \emptyset$, there is nothing changed in $G^{i-1} = (V^{i-1}, E^{i-1})$. This means that the PageRank of all nodes in $G^{i-1}$ remain unchanged. If $\Delta V_{0,i} \neq \emptyset$, and because all nodes in $\Delta V_{0,i}$ are also in $V^{i-1}$, then $\Delta G^i$ may affect some nodes in $G^{i-1}$ through nodes in $\Delta V_{0,i}$. Therefore, we treat nodes in $\Delta V_{0,i} \cup \Delta V_{-,i}$ as the starting nodes to determine all affected nodes in $G^{i-1}$, and update the PageRank of these affected nodes.

To determine the affected nodes in $G^{i-1}$ due to $\Delta G^i$, we consider two factors: one is the distance $\text{dist}(u, v_i)$ between any node $v_i \in V^{i-1}$ and a starting node $u \in \Delta V_{0,i} \cup \Delta V_{-,i}$ and the outdegree at $v_i$, which is $|\text{Out}_{v_i}|$. Similar to wave propagation, we first initialize the impact degree on $u$ due to $\Delta G^i$, which is $\text{aff}_u = 1$; then, the impact degree on $v_i$ is $\text{aff}_v = \alpha e^{-\beta \text{dist}(u, v_i)}$. Moreover, the larger $v_i$'s outdegree is, the more paths it propagates, which means that a smaller impact degree spreads to its outgoing neighbors [35]. Suppose that node $v_{i+1}$ is an outgoing neighbor of $v_i$. Then, $\text{aff}_{v_{i+1}} = \text{aff}_v/|\text{Out}_{v_i}|$, where $|\text{Out}_{v_i}| \neq 0$. Because $|\text{Out}_{v_i}| = 0$, $v_i$ has no any outgoing neighbor, and the impact stops propagating at $v_i$. Thus, the impact degree on each reachable node $v_{i+1}$ from $u$ to $v_{i+1}$ is as follows:

$$\text{aff}_{v_{i+1}} = \begin{cases} 1, & \text{if } v_{i+1} = u \\ \text{aff}_v e^{-\beta \text{dist}(u, v_{i+1})} / \text{Out}_{v_i}, & \text{otherwise}. \end{cases} \quad (6)$$

Let $\delta$ be the threshold to terminate impact propagation, $\delta \in (0, 1)$. If $\text{aff}_{v_{i+1}} < \delta$, the impact degree on $v_{i+1}$ can be ignored. In particular, if $v_{i+1}$ has no any outgoing neighbor, the impact will stop spreading at $v_{i+1}$. Thus, the set of all affected nodes in $G^{i-1}$, denoted as $V^{i}_\text{aff}$, can be obtained as follows:

$$V^{i}_\text{aff} = \{ v \in V^{i-1} | \text{aff}_v \geq \delta \}. \quad (7)$$

To clearly describe the process of finding the affected nodes, we design an algorithm, and the corresponding pseudocode is as follows.

**Proposition 2:** We suppose the distance between two adjacent nodes in a graph is 1, the propagation attenuation coefficient is $\beta$, and the threshold to terminate impact propagation is $\delta$. $G^{i-1}$ is affected by $\Delta G^i$. Then, the length of the farthest path where the impact is propagated in $G^{i-1}$ is $d_{\text{max}} = \left\lfloor \sqrt{1/4 - \ln \delta^2 / \beta - 1/2} \right\rfloor$.

**Proof:** Because $G^{i-1}$ is affected by $\Delta G^i$, we might also assume that the impact starts to propagate from $u$, a reachable node on the propagation path that is denoted as $v_i$, and the outdegree of $v_i$ is $|\text{Out}_{v_i}| \geq 1$. If $\text{Out}_{v_i} = 1$, $i = 1, 2, \ldots$. This impact only propagates along one path, and the distance is the farthest. We suppose this path is $u \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$, where $v_k$ is the farthest reachable node, then according to Formula 6, $\text{aff}_v \times \text{aff}_{v_1} \times \ldots \times \text{aff}_{v_k} = e^{-\beta \text{dist}(u, v_k)} \times e^{-\beta \text{dist}(v_{k-1}, v_k)} \times \ldots \times e^{-\beta \text{dist}(v_1, v_2)} e^{-\beta \text{dist}(v_2, v_1)} \times \ldots e^{-\beta \text{dist}(v_k, v_{k-1})} / \text{Out}_{v_1} \times \ldots / \text{Out}_{v_{k-1}} \times \text{Out}_{v_k} / \text{Out}_{v_{k-1}}$. Since $\text{aff}_u = 1$ and $\text{dist}(u, v_i) = \text{dist}(u, v_{i-1}) + 1$, we have the following: $\text{aff}_{v_i} = e^{-\beta \text{dist}(u, v_i) + \text{dist}(u, v_{i-1})} + \ldots + \text{dist}(u, v_1) + \text{dist}(u, v_{i-1})} + \ldots + \text{dist}(u, v_1) = e^{-\beta (1+2+\ldots+d_{\text{max}})} = e^{-\beta d_{\text{max}}(1+d_{\text{max}})/2}$. According to the assumption of the threshold to terminate propagation, we can obtain $e^{-\beta d_{\text{max}}(1+d_{\text{max}})/2} \geq \delta$, where $d_{\text{max}}$ is the length of the farthest path that the impact is propagated in $G^{i-1}$. Therefore, $d_{\text{max}} = \left\lfloor \sqrt{1/4 - \ln \delta^2 / \beta - 1/2} \right\rfloor$. \hfill $\square$

**Algorithm 1 Finding the Affected Nodes Based on Wave Propagation (FAN_WP)**

**Input:** $G^{i-1} = (V^{i-1}, E^{i-1}), \Delta G^i = (\Delta V^i, \Delta E^i) = (\Delta V_{0,i} + \Delta V_{+,i} + \Delta V_{-,i}, \Delta E_{+,i} + \Delta E_{-,i}), \beta, \delta$

**Output:** $V^{i}_\text{aff}$ // The set of nodes affected by $\Delta G^i$

1. $V^{i}_\text{aff} \leftarrow \emptyset, B V^{i} \leftarrow \emptyset$
2. $V^{i-1} \leftarrow V^{i-1} - \Delta V_{-,i}, E^{i-1} \leftarrow E^{i-1} - \Delta E_{-,i} // Remove the deleted nodes and edges form $G^{i-1}$
3. $G^i \leftarrow (V^{i-1}, E^{i-1})$
4. $B V^{i} \leftarrow \Delta V_{0,i} + \Delta V_{-,i} // Treat the nodes in $\Delta V_{0,i}$ and $\Delta V_{-,i}$ as the boundary nodes
5. if $B V^{i} = \emptyset$ then return $V^{i}_\text{aff}$
6. else for all $v \in B V^{i}$ // Propagate impact from the boundary nodes
7. $\text{aff}_v \leftarrow 1 // Initialize the impact degree on a boundary node
8. $\text{dist}_v \leftarrow 0$
9. end for
10. $V^{i}_\text{aff} \leftarrow B V^{i}$
11. $T m p \leftarrow B V^{i}$
12. while $T m p! = \emptyset$
13. for $V v_i \in T m p$
14. $D_1 \leftarrow \text{descendant}(G^{i-1}, v_i)$ // Get the set of $v_i$'s subsequent nodes
15. if $|D_1| \neq 0$
16. $\text{dist}_{v_i} \leftarrow \text{dist}_{v_i} + 1$
17. for all $v_j \in D_1$
18. $\text{aff}_{v_j} \leftarrow \text{aff}_v e^{-\beta \text{dist}_{v_j}} |D_1| // \text{The impact degree on node } v_j$
19. end for
20. if $\text{aff}_{v_j} \geq \delta$
21. $V^{i}_\text{aff} \leftarrow V^{i}_\text{aff} + \{v_i\} // \text{Add node } v_i \text{ into the set } V^{i}_\text{aff}$
22. end if
23. $D_2 \leftarrow \text{descendant}(G^{i-1}, v_i)$
24. $T m p \leftarrow T m p + D_2$
25. end if
26. end if
27. $T m p \leftarrow T m p - v_i // \text{Remove node } v_i \text{ that have handled from } T m p$
28. end while
29. end if
30. return $V^{i}_\text{aff}$

VOLUME 10, 2022

IEEE Access

15809
V. PAGERANK UPDATE BY AN INCREMENTAL RANDOM WALK

Since a deleted node in a graph does not need to calculate the corresponding PageRank, this section only discusses the update of all affected nodes’ PageRank in the current graph and the calculation of the newly added nodes’ PageRank.

A. UPDATING THE PAGERANK OF THE AFFECTED NODES

1) AN EDGE INSERTION

Suppose that both nodes $u$ and $v$ are the existing nodes in the current graph $G^{t-1}$, but there is no a directed edge $e = (u, v)$. This edge $e$ is newly added to $G^{t-1}$, meaning that there is a new directed edge $e = (u, v)$ in the new graph $G^t$. According to the discussion in section III-B, we can obtain $w_{u,v}^{t-1}$, $r_u^{t-1}$ and $s_{u,v}^{t-1}$ in $G^{t-1}$. To update the PageRank of node $v_i \in V_{aff}^t$, we need to calculate $w_u$, the number of RWPs passing through $u$, and $s_{u,v}$, the total times that RWPs have visited $u$. Following the random walk method in Definition 1, even if the total number of edges in a graph is changed and the total number of nodes in such the graph is not changed, the total number of random walk paths will still not be changed, since $w_{u}^{t} = w_{u}^{t-1}$. The total times of passing through node $u$ can be calculated by Formulas 4 and 5, or $s_{u}^{t} = |w_{u}^{t-1}/(1 - ((1 - \alpha)t_{uv}^{t-1} + Out_{u}^{t-1}r_{u}^{t-1})/(Out_{u}^{t-1} + 1))|$.

Since a new edge is added at node $u$, the total times that all the RWPs have passed through $u$ to subsequent nodes increases. Let $a_u^t$ be the increased number of RWPs passing through $u$ due to adding $e = (u, v)$ as follows:

$$a_u^t = \frac{s_{u}^{t}}{Out_{u}^{t-1}} = \frac{w_{u}^{t-1}}{Out_{u}^{t-1}(1 - \alpha)} + 1. \tag{8}$$

Obviously, adding an edge $e$ may change the total number of times of passing through node $v_i \in V_{aff}^t$, which is denoted as $s_{v_i}^{t-1}$. To calculate $s_{v_i}^{t-1}$ as accurately as possible so as to reduce the random error, we set a larger value $a_u^t$, which is the number of rounds of repeated random walks. Following the random walk method in Definition 1, we perform $a_u^t$ rounds of repeated random walks from node $u$. Once these RWPs start from $u$ and pick $v$ as the next node passes through $v_i$, then $s_{v_i}^{t-1} + 1$. If not, then $s_{v_i}^{t-1} - 1$. After finishing all the specified random walks, node $v_i$ records the updated $s_{v_i}^{t}$, which is $s_{v_i}^{t-1}$ at time $t$. Finally, the PageRank of $v_i \in V_{aff}^t$ can be calculated by using Formula 3.

Since a new edge $e = (u, v)$ is added, the PageRank of the affected node $v_i \in V_{aff}^t$ is changed. To update the PageRank of these affected nodes, we design an algorithm and its pseudocode is as follows.

Proposition 3: For $G^{t-1} = (V^{t-1}, E^{t-1})$, if a new edge $e = (u, v) \notin E^{t-1}$ is added, the set of all nodes in $G^{t-1}$ affected by adding edge $e$ is $V_{aff}^t$. To update the PageRank of the nodes in $V_{aff}^t$, it is stipulated that $a_u^t$-times random walks starting from node $u$ are performed. Compared with the random walk algorithm [9], the minimum computational complexity saved by Algorithm 2 is

Algorithm 2 Updating PageRank After Adding an Edge (URP_AE)

**Input:** $G^{t-1} = (V^{t-1}, E^{t-1})$, $e = (u, v)$, $V_{aff}^t$, $\alpha$, $M$;  
**Output:** The set of the affected nodes’ PageRank $PR_{aff}^t$;  
// $r_{uv}^{t-1}$ is the probability of random walks which start from node $u$ and get back to $u$

1. $PR_{aff}^t \leftarrow \emptyset$, $r_{uv}^{t-1} \leftarrow 0$, $Tmp \leftarrow \emptyset$;
2. for all $v_i \in V_{aff}^t$
3. for $z = 1$ to $M$ // Perform a random walk from any node $v_i$
4. $path_z \leftarrow$ doRandomWalk($G^{t-1}$, $v_i$); // Pass through the specified node $u$
5. if coverSelectedNode($path_z$, $v_i$) = true // Update the number of RWPs passing through $u$
6. $w_u^{t-1} = w_u^{t-1} + 1$;
7. end if
8. end for
9. end for
10. $D_1 \leftarrow$ descendant($G^{t-1}$, $u$); // Get the set of $u$’s subsequent nodes
11. for all $v_i \in D_1$
12. $r_{v_i} \leftarrow \alpha/|D_1|$
13. $Tmp \leftarrow Tmp + (v_i)$.  
14. while $Tmp! = \emptyset$
15. $v_j \leftarrow$ randomChoose($Tmp$); // Get a node randomly
16. $D_2 \leftarrow$ descendant($G^{t-1}$, $v_j$);
17. $r_{v_j} \leftarrow \alpha/|D_2|$
18. $k \leftarrow$ randomChoice($D_2$);
19. if $k \neq u$
20. $Tmp \leftarrow Tmp + \{k\}$;
21. $Tmp \leftarrow Tmp - \{v_i\}$;
22. else
23. $Tmp \leftarrow Tmp - \{v_j\}$;
24. end if
25. $r_{v_i}^{t-1} \leftarrow r_{v_i}^{t-1} + (1 - \alpha)r_{v_j}^{t-1}$;
26. end while
27. end for
28. $s_{v_i}^{t-1} \leftarrow \lfloor w_{u}^{t-1}/(1 - r_{u}^{t-1}) \rfloor$; // The total times that have visited node $u$
29. $E^{t-1} \leftarrow E^{t-1} + \{(u, v)\}$;
30. $a_{u}^{t} \leftarrow \lfloor (w_{u}^{t-1}/|D_1|)(1 - r_{u}^{t-1}) + 1 - (1 - \alpha)r_{v_i}^{t-1}) \rfloor$; // The increased number of RWPs passing through $u$
31. for $z = 1$ to $a_{u}^{t}$ // Perform $a_{u}^{t}$ rounds of repeated random walks from node $u$
32. $path_z \leftarrow$ doRandomWalk($G^{t-1}$, $u$);
33. for all $v_i \in V_{aff}^t$
34. if coverNewEdge($path_z$, $v_i$) = true // Pass through the new edge $(u, v)$
35. $s_{v_i}^{t} \leftarrow s_{v_i}^{t-1} + 1$;
36. $s_{v_i}^{t} \leftarrow s_{v_i}^{t-1}$;
37. else
38. $s_{v_i}^{t} \leftarrow s_{v_i}^{t-1} - 1$;
39. $s_{v_i}^{t} \leftarrow s_{v_i}^{t-1}$;


TRANSLATION: Select the discussion in section VI-A,

\[ \ell \]

 According to Algorithm 2, the computational complexity is

\[ \text{as follows} \]

\[ \] 



\[ \] 

\[ \text{Because an edge} \] 

\[ \text{in the current graph} \]

\[ \text{is deleted} \]

\[ \text{in the new graph} \]

\[ \text{and its pseudocode is as follows.} \]

\[ \text{Algorithm 3 Updating PageRank After Casting Away an Edge (UPR_CE)} \]

\[ \text{Input:} \]

\[ \text{Output:} \]

\[ \text{Algorithm 2} \]

\[ \text{Similarly to Algorithm 2, we omit the details here.} \]

\[ \text{B. CALCULATING THE PAGE RANK OF NEW NODES IN} \]

\[ \text{Generally,} \]

\[ \text{Using the information about adding or deleting nodes or edges contained in} \]

\[ \text{evolve and generate} \]

\[ \text{For these nodes affected by} \]

\[ \text{the PageRank of the affected nodes} \]

\[ \text{Since an edge} \]

\[ \text{is added} \]

\[ \text{is changed. To update the PageRank of these affected nodes, we design an algorithm and its pseudocode is as follows.} \]

\[ \text{Algorithm 3 Updating PageRank After Casting Away an Edge (UPR_CE)} \]

\[ \text{Input:} \]

\[ \text{Output:} \]

\[ \text{Similarly to Algorithm 2, we omit the details here.} \]

\[ \text{B. CALCULATING THE PAGE RANK OF NEW NODES IN } \]

\[ \text{Generally,} \]

\[ \text{Using the information about adding or deleting nodes or edges contained in} \]

\[ \text{evolve and generate} \]

\[ \text{For these nodes affected by} \]

\[ \text{the PageRank of the affected nodes} \]

\[ \text{Since an edge} \]

\[ \text{is added} \]

\[ \text{is changed. To update the PageRank of these affected nodes, we design an algorithm and its pseudocode is as follows.} \]

\[ \text{Algorithm 3 Updating PageRank After Casting Away an Edge (UPR_CE)} \]

\[ \text{Input:} \]

\[ \text{Output:} \]

\[ \text{Similarly to Algorithm 2, we omit the details here.} \]
Algorithm 4 Calculating PageRank in a small Graph with a Super Node (CPR_GSN)

Input: $\Delta G^t = (\Delta V^t, \Delta E^t) = (\Delta V_0^t + \Delta V^{+t} + \Delta V^{-t}, \Delta E^{+t} + \Delta E^{-t}), M$;

Output: The set of new nodes’ PageRank $P_{AV}^t$;

1: for all $u \in \Delta V^{+t}$, do
2: for all $v \in \Delta V_0^t$, do
3: if $\exists e = (u, v)$, do
4: $tmp \leftarrow sn$; // a constant sn as the super node
5: $\Delta E^+ \leftarrow \Delta E^+ - \{(u, v)\}$;
6: $\Delta E^- \leftarrow \Delta E^- + \{(u, tmp)\}$; // Link a new node to the super node
7: end if
8: end for
9: end for
10: for all $(u, v) \in \Delta E^{-+}$, do
11: if $u \in \Delta V_0^t \land v \in \Delta V^t$, do
12: $\Delta E^{+t} \leftarrow \Delta E^{+t} - \{(u, v)\}$; // Remove the edge between two nodes in $\Delta V_0^t$
13: end if
14: end for
15: $E_1^t \leftarrow \Delta E^{+t}$;
16: $V_1^t \leftarrow \Delta V^{+t} + \{sv\}$;
17: $G_1^t \leftarrow (V_1^t, E_1^t)$; // Get the new graph
18: $s_v^t \leftarrow 0$, $v \in V_1^t$;
19: for all $u \in V_1^t$, do
20: for $z = 1$ to $M$
21: $path_z \leftarrow$ doRandomWalk($G_1^t$, $u$); // Perform a random walk from node $u$
22: for all $v \in \Delta V^{+t}$, do
23: if $coverSelectedNode(path_z, v)$ = true // Pass through node $v$
24: $s_v^t \leftarrow s_v^t + 1$;
25: end if
26: end for
27: end for
28: end for
29: for all $v \in \Delta V^{+t}$, do
30: $pr_v^t \leftarrow s_v^t / |V_1^t| |M$; // Calculate the PageRank of any new node $v$
31: $PR_{AV}^t \leftarrow PR_{AV}^t + [pr_v^t]$;
32: end for
33: return $PR_{AV}^t$;

The PageRank $pr_v^t$ corresponding node $v$. Algorithm 4 is used to construct a new graph $G_1^t = (V_1^t, E_1^t)$ and calculate the PageRank $\tilde{pr}_v^t$ corresponding node $v$. Then, the relative error is calculated as follows: $RE = |\tilde{pr}_v^t - pr_v^t| / pr_v^t \leq |V_1^t| / |V^t| - 1$.

Proof: By using Formula 3, we can obtain $pr_v^t = s_v^t / |V_1^t| |M| = w_v^t / (1 - r_v) |V_1^t| |M|$, and $\tilde{pr}_v^t = w_v^t / (1 - r_v) |V_1^t| |M|$ can be obtained by Algorithm 4; Then, the relative error is calculated as follows: $RE = |\tilde{pr}_v^t - pr_v^t| / pr_v^t = |w_v^t / (1 - r_v) |V_1^t| |M| - w_v^t / (1 - r_v) |V_1^t| |M| |V_1^t| |M| / w_v^t / (1 - r_v) |V_1^t| |M|$. According to Algorithm 4, the edges connected to $v$ are not changed during the process of aggregating a supernode. If $M$ is large enough, the probabilities of revisiting $v$ in $G'$ and $G$ are equal, namely, $r_v = r_v'$. Moreover, the number of random walk paths passing through $v$ is $w_v' \leq w_v$. Thus, we obtain $RE \leq |V_1^t| / |V^t| - 1$.

C. Comprehensive Algorithm for All Nodes

Considering a dynamic streaming graph SG, if the local changes occur, the PageRank of all nodes need to be updated in time. Because $G'$ evolves from $G^{-1}$ by using $\Delta G'$, it is reasonable to suppose that the PageRank of all nodes in $G^{-1}$ has been obtained in advance. Moreover, $\Delta G'$ is a small graph including adding or deleting nodes or edges. We consider all the information about $\Delta G'$ and present a comprehensive algorithm, namely an incremental algorithm, to calculate the PageRank of all nodes in $G'$ by using the random walk method. The main process is as follows.

Step 1: When $\Delta G'$ arrives, the set $V_1'$ that includes all the affected nodes in $G^{-1} = (V^{-1}, E^{-1})$ can be obtained by using Algorithm 1. If there is indeed one or more affected nodes, that is, $V_1' \neq \emptyset$, then we move on to Step 2; otherwise, we move on to Step 3.

Step 2: According to $\Delta G' = (\Delta V_0^t + \Delta V^{+t} + \Delta V^{-t}, \Delta E^{+t} + \Delta E^{-t})$, we can determine whether the change is adding or deleting information. If there is an addition case, then we move on to Step 2.1; if there is a deletion case, then we move on to Step 2.2.

Step 2.1: After traversing the set $\Delta E^{+t}$, there are three possible cases: (1) When a new edge $e$ is added between two nodes in $G^{-1}$, the PageRank of nodes in $V_1'$ can be updated by using Algorithm 2. (2) When a new edge $e$ is added between a node in $\Delta V^{+t}$ and a node in $\Delta V_0^t$, then the total number of nodes in $G^{'}$ is $|V^{-1}| + 1$, and the PageRank of nodes in $V_1'$ can be calculated by continuously calling Algorithm 2. (3) In general, if there are many added edges or nodes in $\Delta G'$, then we traverse $\Delta G'$. Step 2 is repeated until the end, and then Step 3 is executed.

Step 2.2: After traversing the set $\Delta E^{-t}$, there are also three possible cases: (1) When an edge $e$ is deleted between two nodes in $G^{-1}$, the PageRank of nodes in $V_1'$ can be updated by using Algorithm 3. (2) When an edge $e$ is deleted due to a deleted node, then the total number of nodes in $G'$ is $|V^{-1}| - 1$, and the PageRank of nodes in $V_1'$ can be calculated by continuously calling Algorithm 3. (3) In general, if there are many deleted edges or nodes in $\Delta G'$, then we traverse $\Delta G'$. Step 2 is repeated until the end, and then Step 3 is executed.

Step 3: For all new nodes, we traverse $\Delta V^{+t}$, call Algorithm 4 to construct a new small graph that includes a supernode, and calculate the PageRank of all new nodes.

Step 4: The PageRank of all nodes in $G'$ has been updated, and all calculation operations are completed.
TABLE 1. The experimental data.

| Name     | Node size | Edge size | Types           |
|----------|-----------|-----------|-----------------|
| wiki-Vote | 7,115     | 103,689   | Directed, Web   |
| amazon0302 | 262,111   | 1,234,877 | Directed, Web   |
| Slashdot0811 | 456,631   | 14,855,875 | Directed, Social |
| RMAT-dg1    | 10,124    | 24,908    | Directed, Synthetic |
| RMAT-dg2    | 170,198   | 359,915   | Directed, Synthetic |

VI. EXPERIMENTAL EVALUATIONS
A. EXPERIMENTAL ENVIRONMENT
To evaluate the effectiveness of our proposed algorithm, we conduct a series of experiments. The hardware environment is a server cluster, including 4 computing nodes connected by high-speed Ethernet. Each computing node has 64-bit Intel(R) Xeon(TM) CPU, 32G RAM and 4T HD. The server cluster runs the Linux operating system Ubuntu 20.04 and the graph processing system GraphX [36]. GraphX is used to store and manage big graphs and to implement parallel and distributed computing. The experimental data we used are: wiki-Vote [37], amazon0302 [38], Slashdot0811 [39], and 2 synthetic datasets. The first three are real-world graphs in application fields, while the last two are synthetic graphs generated by the tool P-MAT [40]. This experimental data is shown in Table 1.

The above graph data is essentially historical static data. However, our research object is the dynamic streaming graph. Therefore, we need to use the above graph data to simulate and generate a dynamic streaming graph. Specifically, 80% of nodes and edges in each graph data are randomly selected for the initial graph. At each time interval $t$, a small scale of nodes is randomly selected as the added or deleted nodes, and a small scale of edges is randomly selected as the added or deleted edges.

B. EXPERIMENTS AND ANALYSIS
We use four different algorithms to conduct our comparative experiments. The first is the power iteration method (PRC_PI) [41], which is a widely used traditional algorithm to calculate PageRank. This algorithm calculates the PageRank of all nodes in a streaming graph starting from the first node, and the result can be treated as the real or true PageRank. The second is an overall random walk algorithm (PRC_RW). The third is UPR_DZIG, which is the state-of-the-art incremental update PageRank algorithm implemented in [20]. The fourth is our algorithm, which is an incremental random walk algorithm (UPR_IRW).

1) ACCURACY
   Experiment 1: The comparison between the overall random walk algorithm PRC_RW and the traditional power iteration algorithm PRC_PI

   To evaluate the accuracy of PRC_RW, we set different numbers of rounds in random walks, $M = 2, 4, 6, 8, 10, 15, 20, 25, 30, 35$, and execute these two algorithms over 5 graph datasets. The results of PRC_RW are compared with the results of PRC_PI, which is regarded as the real PageRank. The mean relative error MRE and the standard deviation SD are calculated. The experimental results are shown in Figs. 4 and 5. As we can see, the MRE over wiki-Vote becomes small and stable when $M$ is larger than 4 and the SD tends to be stable when $M = 10$. Likewise, the MRE over amazon0302 becomes small and stable when $M$ is larger than 15, and the SD tends to be small and stable when $M = 6$. Both MRE and SD over Slashdot0811 become small and stable when $M$ is larger than 20. For RMAT-dg1 and RMAT-dg2, their MREs become small and stable when $M$ is larger than 6, and their SDs tend to be stable when $M = 20$. Therefore, as long as we are setting a reasonable $M$, we can obtain smaller and relatively stable MRE and SD, meaning that the calculated PageRank is more accurate. Moreover, as $M$ increases, MRE decreases and converges to a smaller error range. This is consistent with the conclusion of Proposition 1 in Section III-B.

   Experiment 2: The comparison between our incremental random walk algorithm UPR_IRW and the traditional power iteration algorithm PRC_PI

   First, we set the threshold to terminate impact propagation with different values, $\delta = 0.1, 0.2, \ldots, 0.9$. Experiments are carried out separately. We observe the proportion of the affected nodes to all nodes, and compare the mean relative error MRE between UPR_IRW and PRC_PI.
The experimental results are shown in Fig. 6(a) and (b). In general, the larger \( \delta \) is, the smaller the affected area is. As we can see in Fig. 6(b), almost all curves show an upward trend. This means that the affected nodes in the graph cannot be completely found as \( \delta \) increases, and thus, the MRE becomes larger. According to the above experimental results, \( \delta = 0.5 \) is a reasonable choice.

To analyze the impact of adding or deleting different sizes of nodes on updating PageRank, we randomly add or delete nodes with different proportions from each graph dataset in Table 1, \( \rho = 0.1\%, 0.5\%, 1.0\%, 2.0\%, 4.0\%, 6.0\%, 8.0\%, 10.0\%, 15.0\% \) and \( 20.0\% \). We use UPR_IRW and PRC_PI to calculate the PageRank of all nodes in the graph, and further calculate the mean relative error MRE. The experimental results are shown in Fig. 7(a) and (b). With the increasing proportion of adding or deleting nodes, the corresponding MRE increases to a certain extent. This is because the graph data may have a mutation, which leads to a decrease in the accuracy of our algorithm UPR_IRW.

**Experiment 3:** The comparison between incremental random walk algorithm UPR_IRW and the overall random walk algorithm PRC_RW

We set the sizes of new nodes to 1.0\%, 2.0\%, 3.0\%, 4.0\% and 5.0\% relative to the maximum size of each graph dataset, and the threshold to terminate impact propagation is \( \delta = 0.5 \). We use PRC_RW and our algorithm UPR_IRW to calculate the PageRank of new nodes, and then obtain the corresponding average relative error MRE. The experimental results are shown in Fig. 8. As we can see, MRE is small and less than 0.01. If the proportion of new nodes is small, the MRE is close to 0. Therefore, the effect of the incremental random walk algorithm UPR_IRW is very similar to that of the overall random walk algorithm PRC_RW.

In addition, we treat these 5 graph datasets as the initial graph and randomly add 10 new nodes. We use UPR_IRW and PRC_RW to calculate the PageRank of these 10 new nodes respectively, and then obtain the relative error corresponding to each node. The experimental results are shown in Fig. 9. As we can see, the relative error of each new node is less than the maximum relative error, which is consistent with the conclusion of Proposition 4.

**Experiment 4:** The comparison between the incremental random walk algorithm UPR_IRW and the state-of-the-art incremental computing algorithm UPR_DZIG.
FIGURE 9. A comparison between the relative error and the maximum relative error.

FIGURE 10. MRE of UPR_IRW and UPR_DZIG compared with PRC_PI.

We set the sizes of new nodes to 1.0%, 2.0%, 3.0%, 4.0% and 5.0% relative to the maximum size of each graph dataset, and the threshold to terminate impact propagation is $\delta = 0.5$. We use UPR_DZIG and our algorithm UPR_IRW to calculate the PageRank of new nodes, and then compare the results with the real PageRank to obtain the corresponding average relative errors, respectively. The experimental results are shown in Fig. 10. As we can see, MRE is small and less than 0.01. For the graph dataset amazon0302, the MREs of UPR_DZIG are slightly smaller than the MREs of UPR_IRW. However, UPR_IRW performs better than UPR_DZIG on these graph datasets.

2) EFFICIENCY

Experiment 5: The speedup comparison between the incremental random walk algorithm UPR_IRW and the traditional power iteration algorithm PRC_PI

We set the sizes of new nodes to 0.1%, 0.2%, 0.5%, 1.0%, 2.0%, 3.0%, 4.0%, 5.0%, 6.0% and 8.0% relative to the maximum size of each graph dataset, and the threshold to terminate impact propagation is $\delta = 0.5$. We use UPR_IRW and PRC_PI to calculate the PageRank of all the nodes in each graph. Suppose that the execution times of UPR_IRW and PRC_PI are $T(UPR\_IRW)$ and $T(PRC\_PI)$ respectively, and the speedup of UPR_IRW to PRC_PI is $Speedup = \frac{T(PRC\_PI)}{T(UPR\_IRW)}$. The experimental results are shown in Fig. 11. Compared with PRC_PI, our algorithm UPR_IRW accelerates the speed of computing PageRank tremendously, especially in graph dataset RMAT-dg2, and the speedup can be as high as 99.50. As the size of new nodes increases, the speedup decreases gradually. Anyway, the speedup is much greater than 1, which indicates that the incremental random walk algorithm UPR_IRW is faster than the traditional power iteration algorithm PRC_PI.

Experiment 6: The speedup comparison between the incremental random walk algorithm UPR_IRW and the overall random walk algorithm PRC_RW

We use the same experimental environment and parameters stated in Experiment 5, and repeatedly execute UPR_IRW and PRC_RW to calculate the PageRank of all nodes in the graph. In addition, the speedup of UPR_IRW to PRC_RW is $Speedup = \frac{T(PRC\_RW)}{T(UPR\_IRW)}$. The experimental results are shown in Fig. 12. Compared with PRC_RW, our algorithm UPR_IRW greatly accelerates the speed of calculating PageRank, especially on the graph dataset RMAT-dg1, and the speedup can be as high as 82.62. As the size of new nodes increases, the speedup decreases gradually. The speedup is much greater than 1, which indicates that the incremental random walk algorithm UPR_IRW is faster than the overall random walk algorithm PRC_RW.

Experiment 7: The speedup comparison between the incremental random walk algorithm UPR_IRW and the state-of-the-art incremental computing algorithm UPR_DZIG

We use the same experimental environment and parameters stated in Experiment 5, and repeatedly execute UPR_IRW and UPR_DZIG to calculate the PageRank of all nodes in the graph. In addition, the speedup of UPR_IRW to UPR_DZIG is $Speedup = \frac{T(UPR\_DZIG)}{T(UPR\_IRW)}$. The experimental results are shown in Fig. 13. If the size of new nodes is smaller than 1.0% on these two graph datasets
on the aggregation idea is designed, which can accurately calculate the PageRank of these new nodes. In summary, a comprehensive algorithm based on incremental random walks for updating the PageRank in a streaming graph is proposed, and a deep analysis is given. Finally, we use the proposed algorithm, the traditional power iteration algorithm and the overall random walk algorithm to conduct a series of comparative experiments on 3 real-world graphs and 2 synthetic graphs. The experimental results show that our algorithm has better capability in terms of the speed, while maintaining the accuracy. In future work, we will prove the consistency of calculating PageRank between our incremental computation method and the overall computation method through theoretical derivation. Moreover, we will transplant the proposed algorithm to some open-source big graph computing systems and evaluate its effectiveness over more real-world graph datasets.

REFERENCES

[1] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” Comput. Netw. ISDN Syst., vol. 30, nos. 1–7, pp. 107–117, Apr. 1998.
[2] R. Cheng, E. Chen, J. Hong, A. KyrOLA, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, and F. Zhao, “Kineograph: Taking the pulse of a fast-changing and connected world,” in Proc. 7th ACM Eur. Conf. Comput. Syst. (EuroSys), 2012, pp. 85–98.
[3] J. Riely, “Updating PageRank for streaming graphs,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW), May 2016, pp. 877–884.
[4] K. Liubonko and D. Sávez-Trumper, “Matching Ukrainian Wikipedia red links with English Wikipedia’s articles,” in Proc. Web Conf., Apr. 2020, pp. 819–826.
[5] A. N. Langville and C. D. Meyer, “Updating Markov chains with an eye on Google’s PageRank,” SIAM J. Matrix Anal. Appl., vol. 27, no. 4, pp. 968–987, Jan. 2006.
[6] W. Fan, X. Wang, and Y. Wu, “Incremental graph pattern matching,” ACM Trans. Database Syst., vol. 38, no. 3, pp. 1–47, Aug. 2013.
[7] H. Zhao, X. Xu, Y. Song, D. L. Lee, Z. Chen, and H. Gao, “Ranking users in social networks with motif-based PageRank,” IEEE Trans. Knowl. Data Eng., vol. 33, no. 5, pp. 2179–2192, May 2021.
[8] A. D. Sarma, S. Gollapudi, and R. Panigrahy, “Estimating PageRank on graph streams,” J. ACM, vol. 58, no. 3, pp. 1–19, May 2011.
[9] Z. Zhan, R. Hu, X. Gao, and N. Huai, “Fast incremental PageRank on dynamic networks,” in Proc. Web Eng., 2019, pp. 154–168.
[10] P. Desikan, N. Pathak, J. Sivastava, and V. Kumar, “Incremental page rank computation on evolving graphs,” in Proc. 14th Int. Conf. World Wide Web (WWW), 2005, pp. 1094–1095.
[11] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, “Extrapolation methods for accelerating PageRank computations,” in Proc. 12th Int. Conf. World Wide Web (WWW), 2003, pp. 261–270.
[12] J. Arnal, H. Miguéllon, V. Migallón, J. A. Palomino, and J. Penadés, “Parallel relaxed and extrapolated algorithms for computing PageRank,” J. Supercomput., vol. 70, no. 2, pp. 637–648, Feb. 2014.
[13] B. Nigam, S. Tokekar, and S. Jain, “Predicting the next accessed web page using Markov model and PageRank,” Int. J. Data Mining Emerg. Technol., vol. 3, no. 2, pp. 73–80, 2013.
[14] T. Fushimi, K. Saito, K. Ohara, M. Kimura, and H. Motoda, “Efficient computing of PageRank scores on exact expected transition matrix of large uncertain graph,” in Proc. IEEE Int. Conf. Big Data (Big Data), Dec. 2020, pp. 916–923.
[15] K. Christian, P. A. Chirita, and W. Nejdl, “Efficient parallel computation of PageRank,” in Proc. Eur. Conf. Inf. Retr., 2006, pp. 241–252.
[16] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, “PowerGraph: Distributed graph-parallel computation on natural graphs,” in Proc. USENIX Conf. Oper. Syst. Design Implement., 2012, pp. 17–30.
[17] B. Bahmani, A. Chowdhury, and A. Goel, “Fast incremental and personalized PageRank,” Proc. VLDB Endowment, vol. 4, no. 3, pp. 173–184, 2010.
ZHIPENG SUN was born in 1990. He received the M.S. degree in computer application technology from the Department of Computer Internet of Things Engineering, Jiangnan University, in 2017. He is currently pursuing the Ph.D. degree in computer software and theory with Tongji University. His research interests include big data analysis, graph computing, machine learning, and artificial intelligence.

GUOSUN ZENG (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer software and application from the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He is currently working at Tongji University, as a Full Professor, and as a Ph.D. Supervisor of computer software and theory. His research interests include cloud computing, big data processing, software verification, artificial intelligence, parallel computing, and information security.

CHUNLING DING received the B.S. degree from the Huazhong University of Science and Technology and the M.S. degree from Tongji University. She is currently working as a Senior Engineer with Tongji University. Her main research interests include data analysis, computer simulation, and computer applications.

* * *