Finding Top-*k* Answers in Node Proximity Search Using Distribution State Transition Graph

Jaehui Park and Sang-Goo Lee

Considerable attention has been given to processing graph data in recent years. An efficient method for computing the node proximity is one of the most challenging problems for many applications such as recommendation systems and social networks. Regarding large-scale, mutable datasets and user queries, top-*k* query processing has gained significant interest. This paper presents a novel method to find top-*k* answers in a node proximity search based on the well-known measure, Personalized PageRank (PPR). First, we introduce a distribution state transition graph (DSTG) to depict iterative steps for solving the PPR equation. Second, we propose a weight distribution model of a DSTG to capture the states of intermediate PPR scores and their distribution. Using a DSTG, we can selectively follow and compare multiple random paths with different lengths to find the most promising nodes. Moreover, we prove that the results of our method are equivalent to the PPR results. Comparative performance studies using two real datasets clearly show that our method is practical and accurate.

Keywords: Personalized PageRank, Top-*k* Node Proximity Search, Path Selection Algorithm, Distribution State Transition Graph, Graph Search.

Manuscript received Mar. 11, 2015; revised Mar. 15, 2016; accepted Apr. 4, 2016.

This work was supported by the Knowledge Services Industry Core Technology Development Program (10051028, Development of Predictive Manufacturing System using Data Analysis of 4M Data in Small and Medium Enterprises) funded By the Ministry of Trade, Industry & Energy (MI, Korea).

Jaehui Park (corresponding author, jaehui@etri.re.kr) is with the SW & Contents Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Sang-Goo Lee (sglee@snu.ac.kr) is with the Computer Science and Engineering Department, Seoul National University, Rep. of Korea.

I. Introduction

1. Motivation

A graph is a fundamental data structure for capturing relationships among entities, and may represent useful abstractions of real-world objects. Computing graph data has become important with the increasing needs of recent applications, such as social networks, online ads/spam networks, and protein interaction networks. Traditional studies have covered many problems regarding the use of graphs, for example, shortest paths and subgraph isomorphism. One of the most well-known problems is computing the node proximity. The node proximity can be applied, for example, to finding potential friends for a person based on their social network. Many proximity measures and their computation algorithms have been proposed in the literature [1]–[7]. Based on such studies, various applications have emerged, including name disambiguation [1], spam filtering [2], and friend recommendations in social networks [3], and others [8], [9].

In this paper, we consider a node proximity measure called Personalized PageRank (PPR) [2], which is one of the most well-known graph metrics owing to its utility and solid theoretical foundation. PPR is defined as the stationary distribution of random walks; at each step, it follows an outgoing edge from the current node with a certain probability, or it jumps to a certain node in accordance with a given preference distribution. For more information, refer to [4]. However, the computing of PPR poses challenging issues [4], [10]–[15]. As an example, a power iteration is known to be a naïve computation method for PPR [4], which converges very slowly, and the number of iteration steps is excessive for large graphs. To remedy such inefficiency, several studies have

http://dx.doi.org/10.4218/etrij.16.0115.0229

been conducted, including pre-computations [16], matrix partitioning [5], and Monte Carlo approximations [11]. However, regarding large-scale, mutable datasets and queries in recent applications, existing studies may not work efficiently. In some applications, users often want to know the top-most answers more quickly while ignoring the others. Hence, in this paper, we try to solve the problem of finding top-k proximity nodes using the PPR measure. Many top-k approaches have been introduced in the literature [12], [17], [18]. For more details, refer to the related works section. The difference between our approach and existing studies is that, not only do we reduce the computation time for producing top-k answers, we also propose a novel and useful computation model for further extensions, such as parallel processing.

2. Our Approach

The problem dealt with in this paper is finding top-k nodes with the highest PPR scores, not computing the actual scores of an entire node set of a given graph. It would be useful to know the potentials of the nodes at different iteration stages before their convergence. By comparing such potentials, for example, the probabilities, we can determine the top-k answers without computing the PPR equation, which is conducted using a sparse matrix-vector multiplication in previous studies. Our approach works as follows: 1) identifying every possible move of a random surfer at every iteration to compute the PPR equation, 2) selectively following the paths to note the probabilities assigned to the nodes at each iteration, 3) determining promising nodes based on the state of the probability distribution, and 4) terminating when we guarantee that the distribution state no longer changes and we have k promising nodes. However, this idea cannot be easily achieved using conventional PPR computation models. Therefore, we introduce a conceptual structure, a distribution state transition graph (DSTG), to depict the iterative steps for computing the complete PPR vector, and determine a random surfer's paths in reaching the promising candidates. Based on a DSTG, we can compute the distribution state of a PPR vector of candidate nodes at different iteration stages. To highlight this, we avoid the computational cost of whole matrix-vector multiplications, while maintaining only the top-k promising nodes. In particular, we developed a greedy algorithm, the path selection algorithm (PSA), to select nodes having a higher probability to be PPR scores by tracking paths with different lengths. The partial set of promising nodes and their different iteration levels are involved in the PPR computation, and the approach dramatically decreases the computational time. Herein, we provide a theoretical analysis showing that our algorithm produces equivalent results to exact answers, and that in practice it is better than existing algorithms. To do so, we conducted experiments on two real-world datasets.

The contributions of this paper are summarized as follows:

- We introduce a DSTG to depict the iterative steps of computing the PPR scores, and determine the path reaching the promising nodes.
- We introduce a greedy algorithm to test the candidate nodes to be included in the top-k list based on the accumulated weights of the intermediate PPR scores. To highlight, it avoids the computational cost of matrix-vector multiplications.
- We provide a theoretical analysis showing that our algorithm produces results equivalent to those of existing studies, and that in practice our algorithm is better than existing algorithms.

3. Organization

The rest of this paper is structured as follows. In Section II, we describe the basic notions and related works. In Section III, we introduce a novel model, DSTG, and a top-k processing algorithm, PSA. In Section IV, we report the experimental results. Finally, some concluding remarks and areas of future work are provided in Section V.

II. Background

1. Proximity measure: Personalized PageRank

To describe our idea, we briefly outline the basic concepts used throughout this paper. Given a weighted directed graph $G = (V, E)$ with n nodes, $V(G)$, and m edges, $E(G)$, the weight on each edge, $e(u, v) \in E(G)$, is denoted by probability w. For the sake of simplifying the presentation of some of the formulae, assume for the rest of the paper that the edge weights w on the outgoing edges $O(u)$ of each node u will be uniformly distributed, that is, $1/|O(u)| = w$. PageRank [19] is one of the best-known measures of importance of graph nodes represented as Web pages. Assume a random surfer moves from page u to another page v with a uniform probability of $1/|O(u)|$. PageRank corresponds to a probability distribution of a random surfer's existence in a set of pages under a steady state. PageRank adopts the structural information of Web graphs to effectively quantify the global importance of Web pages. The fact that a Markov chain approximates PageRank using reasonable constraints makes the measure useable as a theoretical foundation for various applications. In this paper, we focus on adding query-specific factors to the metric.

We investigated its variant, PPR, which considers a given node. At the initial stage, once a random surfer is located in node s, it iteratively moves to one of its neighbors with a certain probability, which sums up to α (transition probability)

or jumps to the original node s with a probability of $1 - \alpha$ (teleport probability). PPR is the same as PageRank, except for the jumps, which can be designed as a probabilistic distribution of the user's preference over the entire nodes. As in the probabilistic tendency of returning moves to the user's (given) preferential nodes s, this measure extends PageRank to a "personalization." We refer to node s as a *start node*, which can be multiple nodes represented as vector $\vec{s}$, for example, a *teleport vector*. Let $\vec{s}$ be an $n \times 1$ column-normalized vector of the preference distribution, that is, $\sum_i^n \vec{s}(i) = 1$, where $\vec{s}(i)$ denotes the i-th elements of vector $\vec{s}$, and W be an $n \times n$ column normalized adjacency matrix, which is referred to as a transition matrix, of graph G. Therefore, given the distribution $\vec{s}$ and matrix W, the PPR score can be defined as a recurrence equation as follows:

$$\vec{p}_s^{(t)} = \alpha \cdot W \vec{p}_s^{(t-1)} + (1-\alpha) \cdot \vec{s}. \quad (1)$$

For our purpose, we can use t as a time unit that describes the subsequent steps of a random surfer's movements. Equation (1) illustrates the probability distribution of the random surfer's location after t steps. As a result, the vector $\vec{p}_s^{(t)}$ will be an $n \times 1$ vector, where n is the number of nodes, and the i-th element $\vec{p}_s^{(t)}(i)$ denotes the probability that the random surfer exists at node i. The value $\vec{p}_s^{(t)}$ will be converged if t becomes much larger. The convergence of the equation is guaranteed, as proved in [20].

2. Related Works

To compute the exact PPR score in a straightforward way, the power iteration method updates $\vec{p}_s^{(t)}$ recursively in (1) until the convergence is reached efficiently computing matrix is key techniques to other areas, such as [21]. However, the repetitive matrix-vector multiplication incurs an excessive computational cost for large graphs. There are many previous studies on alleviating the processing cost of the PPR computation. As a seminal work, Jeh and Widom [4] compute and store the scores of the chosen nodes. PPR scores can be computed using a linear combination of pre-computed scores. A linear algebraic solution [22] introduces the state of convergence by deducing (1) to consider the infinite time ($t \approx t - 1$), as in the following equation:

$$\vec{p}_s = \alpha \cdot W \vec{p}_s + (1-\alpha) \cdot \vec{s}. \quad (2)$$

Equation (2) can be solved if its inverse matrix exists (proved in [4]), as follows:

$$\vec{p}_s = (1-\alpha)(I - \alpha \cdot W)^{-1} \cdot \vec{s}. \quad (3)$$

If we obtain the inverse matrix $(I - \alpha \cdot W)^{-1}$ in (3) at the pre-computation time, $\vec{p}_s$ can be computed online. However, this method also requires a quadratic space for storing the inverse matrix. Although the approach is faster than the original approach, the convergence of the linear algebraic approach is not stationary.

The Monte Carlo approaches [11], [18] assume that a number of random surfers move from a given start node and stop according to a geometric distribution. The Monte Carlo approaches can approximately compute the top-k nodes in an *ad-hoc* style because they perform random walks on a given graph on the fly. The approach in [18] applies the Monte Carlo method to mathematically conclude that the number of random walks has a Poisson distribution, and the expected number of random walks can be evaluated with a given error bound. However, a Monte Carlo approach requires the number of random walks to be set, which induces a trade-off between efficiency and approximation quality. After repetitive runs, the PPR scores can be evaluated for nodes based on the number of random surfers in them. Bahmani and others [10] proposed a disk-based approach, which may consume high I/O operations. The study [23] presented a local algorithm that computes the PPR answers by adaptively considering a small set of nodes near a given node. The method in [5] reduces the storage cost for a matrix inversion. It divides the transition matrix into several small matrices using graph partitioning, and pre-computes the inversions of smaller matrices.

Instead of computing the entire PPR scores, there are some researches that have concentrated on finding the top-k nodes efficiently. They only require the top-k values $\{\vec{p}_s^{(t)}(i) | 1 \le i \le n, 1 \le t\}$ for given query $\vec{s}$, which is referred to as the top-k PPR computation problem. The approach in [13] modifies the bookmark coloring algorithm [17] to answer top-k queries in an entity-relation graph. The phenomenon of propagation of a coloring substance is observed in such graphs. During the computation of the PPR scores, this method compares the current k-th value with the upper bound of the $(k + 1)$-th value, and terminates the process when the top-k nodes are completely determined. These methods, however, are restricted to either a specific graph model or a theoretical point of view, and thus are difficult to be used in real problems. The k-dash algorithm (KDA) [12], [24], which is a state-of-the-art algorithm, was proposed with two main ideas, a sparse matrix computation and a tree estimation. The work in [24] is based on the same rooted work of [12], and thus we consider the two algorithms to be the same (as in KDA) with minor variants. KDA first reorders nodes to make the inverse matrix from (2) sparse during the pre-computation phase, and constructs a breadth-first search tree rooted at the given node.

KDA efficiently and precisely finds the top-k highest proximity nodes by computing the proximities of the nodes in ascending order of distance from the root node. The authors of [25] identify a uniform property over many PPR measures in which each node has at least one adjacent node having a lower (or higher) proximity. This relationship is utilized to find the top-k nodes satisfying the no-local-optimum property. A novel problem, reverse top-k proximity, is proposed in [26] to suggest an another measure for understanding the distance between nodes in a graph.

3. Problem Statement

In this paper, we consider the problem of a top-k PPR computation: Given a weighted directed graph $G = (V, E)$ and teleport vector $\vec{s}$, find the k nodes with the largest values in the PPR score vector $\vec{p}_s$. A PPR score is the steady-state probability of a node computed by (1). Under a steady state, the probability distribution over the entire set of nodes determines the final order of the nodes. Because we focus on the top-k nodes, we only need the probability distribution of a partial set of nodes sufficient to be on the final list, instead of computing all nodes. To illustrate our method, we transform the problem as follows:

Problem Definition. Given a graph $G = (V, E)$, a teleport vector $\vec{s}$, a restart probability parameter α, and a constant k, find k nodes whose scores in vector $\vec{p}_s^{(t)}$ are expected to be the largest with respect to the nodes in $\vec{s}$.

III. Top-k Personalized PageRank with Distribution State Transition Graph

1. Data Model

Our idea starts from observing the changing probability distribution over vector $\vec{s}$, and then trying to find the most promising nodes at early stage t. To do so, we need to trace the changing states of the probability distribution at each recursion of (1). However, it is difficult to identify the nodes to eventually be in the final list, and detect the right time to stop the recursive operations. We translate the recursive equation to a set of iterative equations that can identify a random surfer's movements and corresponding changes in the distribution states. As a visible interpretation of the iteration results and changes, we use a trellis graph. A trellis graph is a directed graph with nodes and directed edges that satisfy the following conditions: 1) the node set is partitioned into subsets $L = \{l_0, l_2, \ldots, l_n\}$, and 2) the edges connect nodes only of consecutive subsets l_t, and l_{t+1}, for $0 \le t \le n$. A trellis graph is useful for representing the subsets as an ordered time sequence to formulate many problems arising in the field, such as in communications or information theory. Figure 1 illustrates an example of a trellis graph with six subsets. Based on the structure, we define a novel data model, DSTG, with useful properties for representing the PPR semantics.

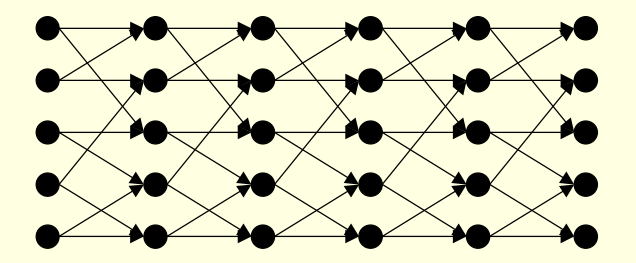

Fig. 1. Trellis graph with $|L| = 6$ and $|l_i| = 5$.

Definition 1. (distribution state transition graph) With respect to graph $G = (V, E)$, DSTG $\mathrm{D}^G(n, L)$ is a modified trellis graph satisfying the following properties:

1) **State:** There exists a one-to-one and onto function from each state l_t $(1 \le t \le n)$ of D^G to $V(G)$. Every node $v_i^{(t)}$ in l_t corresponds to $v_i \in V(G)$.
2) **Transition:** An edge $e_{ij}(v_i^{(t)}, v_j^{(t+1)})$ exists from node $v_i^{(t)}$ to nodes $v_j^{(t+1)}$ if the corresponding edge $e_{ij}(v_i, v_j) \in E(G)$ exists.
3) **Distribution:** An edge $e_{ii}(v_i^{(t)}, v_i^{(0)})$ exists from node $v_i^{(t)}$ to nodes $v_i^{(0)}$. There exists a one-to-one and onto function from state l_0 of D^G to $V(G)$. There are no outgoing edges from the nodes in l_0.

A DSTG depicts a stochastic process of a random surfer's location over node set $V(G)$ at a certain time t using the *state* property. The probabilities of the surfer's movements are described by the properties, *transition* and *distribution*. To simplify the notations, we use the same labels on the nodes of graph G and subset l_t $(0 \le t \le n)$ in D^G. At a certain time t, the edges directing to the nodes in the next state l_{t+1} describe the possibility of a transition, and those to the nodes in the zeroth state l_0 describe the possibility of a jump. Figure 2 illustrates an example graph G and its corresponding DSTG. A probabilistic path $p \rightarrow q \rightarrow r$ of a random surfer is denoted by a double red line in both graphs, G and D^G. The edges with the transition property are denoted by the solid lines. The edges with the distribution property to state l_0 are denoted by a dotted line only for the top nodes, and those for the other nodes are omitted for a simple presentation. As a result, all probabilistic movements of a random surfer are modeled in DSTG iteratively according to the subsequent states L. We consider state l_0 as a special case for the surfer's movements. Because the nodes in state l_0 have

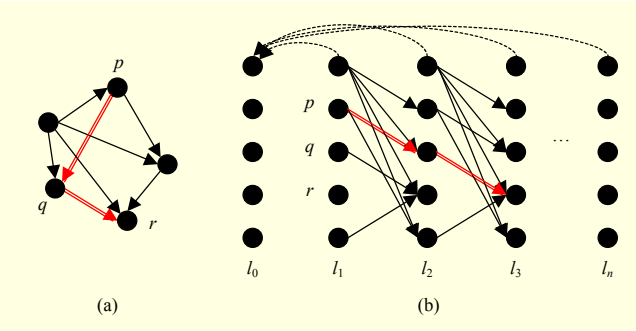

Fig. 2. (a) Directed graph G with a random path $p \rightarrow q \rightarrow r$, and (b) corresponding DSTG, D^G.

no outgoing edges, the surfer always stays in a special state once they jump in.

2. Weight Distribution Model

In the previous section, we proposed a novel model, the DSTG, to depict the probabilistic movements of a random surfer on a graph. In this section, we try to explain the weight assignment model that represents the probability of existence of a random surfer on node v_i at time t. Based on the proposed model, we can discuss its equality to the PPR score.

Now, we introduce computational factors to represent the weights for the nodes and edges in a DSTG. The node weights represent the probability of arrival at a certain node at a certain time. The edge weights represent the probability of departure from a certain node at a certain time. As we know, state l_0 has no outgoing edges, and represents the terminal state of a random surfer's movements. It is worth noting that an update of the node weights at state l_0 during the processing of a DSTG is similar to the recurrence of solving (1). We introduce a weight distribution model of D^G as follows:

Definition 2. The weight distribution model of D^G is defined to compute the probabilistic distribution of a random surfer's existence in node set $V(G)$ of G at a certain time t. This probabilistic distribution explains the approximation of the PPR score vector $\vec{p}_s^{(t)}$. With respect to (1), the model is defined as follows:

1) **Edge Weight:** Given parameter α, an outgoing edge e_{ij} from nodes $v_i^{(t)}$ in state l_t, the edge weight $W(e_{ij}(v_i^{(t)}, v_j^{(t+1)}))$ is assigned as $\alpha \times W(e_{ij}(v_i, v_j))$ of G. As a special case, that is, an incoming edge e_{ii} to nodes $v_i^{(0)}$ in state l_0, the edge weight $W(e_{ii}(v_i^{(t)}, v_i^{(0)}))$ is assigned as $1 - \alpha$.
2) **Node Weight:** For every node $v_i^{(t)}$ in l_t, the node weight $W(v_i^{(t)})$ is assigned as

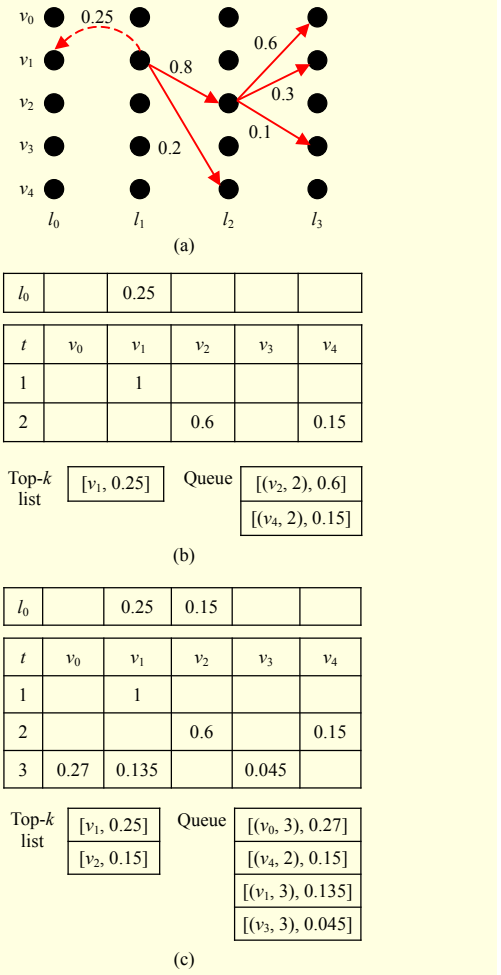

Fig. 3. (a) Random paths starting from v_1, and (b) the first and (c) second runs of the PSA.

$$\sum_j W(v_j^{(t-1)}) \times W\left(e_{ji}(v_j^{(t-1)}, v_i^{(t)})\right), \tag{4}$$

and at the same time, node weight $W(v_i^{(0)})$ is assigned as

$$W(v_i^{(0)}) + W\left(e_{ii}(v_i^{(t)}, v_i^{(0)})\right). \tag{5}$$

In this model, the weights of the edges are assigned to consider the transition probability with a given parameter α and the teleport probability with $1 - \alpha$. Similar to the original problem of the PPR score computation, the edge weights are typically given as a parameter, for example, a transition matrix. However, the node weights should represent the chronological order, as in (4), of random movements based on the modified trellis graph form D^G. Each subset l_t represents the probabilistic distribution state of a certain time t, and its node weight $W(v_i^{(t-1)})$ should be computed before computing $W(v_j^{(t)})$. Moreover, equation (5) describes the updating weight for the nodes in the zeroth state by $W(e_{ii}(v_i^{(t)}, v_i^{(0)}))$ at each move.

Owing to the probabilistic settings, the sum of the weights $\sum_i W(v_i^{(t)})$ reach closer to zero, and $\sum_i W(v_i^{(0)})$ reaches closer to 1. Figure 3(a) illustrates an example of random moves starting from node v_1. The edge with the distribution property is denoted by a dotted line. When we consider the steps from $v_1^{(1)}$ to the next state l_1, state l_0 is updated at the same time. As a result, the node weights are updated in the time-vertex tables in Fig. 3. State l_0 accumulates the visiting probability of the nodes. Detailed examples in both Figs. 3(b) and 3(c) are described in the next section. By computing node weights in a DSTG, we can derive the stationary distribution. Noting that we construct a DSTG and a weight distribution model to interpret the random surfer's movements, the distribution state can approximate the PPR. To make our interpretation more concrete, we prove that the weight distribution in l_0 is identical to the PPR score vector. The following theorem claims the equality through a proof.

Theorem 1. The probability distribution of a random surfer at state l_0 at time t is identical to PPR score vector $\vec{p}_s^{(t)}$, where $t \approx \infty$.

Proof. Assume that a random surfer begins at state l_1 with a given start distribution. The surfer can move to the second state with probability α or can move to state l_0 with a probability of $1 - \alpha$. If the surfer moves to the third state, the probabilistic move is to the next state or state l_0 with probability α or $1 - \alpha$, respectively. The same process is repeated each time. If the surfer moves to state l_0, there are no outgoing edges. Hence, the probability distribution of the surfer arriving at each node in state l_0 is determined at every intermediate state l_t $(0 < t < n)$. If we use $P(v_i^{(t)})$ to represent the probability of a random surfer arriving at node v_i at time t, the probability of the surfer arriving at node v_i in state l_0, $P(v_i^{(0)})$, is computed as follows:

$$P(v_i^{(0)}) = \sum_i^{\infty} P(v_i^{(i)}) \times (1-\alpha)\alpha^{i-1}. \quad (6)$$

Because (6) is identical to the *inverse P-distance* [4], if a jump to state l_0 is regarded as a stop, it is concluded that the stationary distribution of a random surfer in state l_0 is exactly the same with the PPR. ■

Now, we note that the problem we defined in section 2.3 can be solved by traversing a DSTG. Based on Theorem 1, we can obtain the exact PPR scores by accumulating the weight $W(v_i^{(0)})$. In other words, we can observe the changes in distribution state in l_0 for all moves of the surfer. Actually, traversing an entire DSTG is infeasible because the size n is infinite. However, the advantage of using a DSTG is that we can compare the probabilities at different times t and $t + c$. Because this model presents every separate path at every state, it allows a search algorithm to easily determine which paths are promising. Moreover, we can proceed with more steps on some paths having a high potential and defer some unpromising paths. Now, we need an algorithm to determine when random moves stop because we are confident that the distribution state no longer changes. To be more specific, the order of some nodes in l_0 with respect to the node weights undergoes no further changes. We denote "some" nodes as top-k nodes.

Algorithm 1. Path Selection Algorithm.

Input**:** graph $G(V, E)$, teleport vector r, constant eps, a
Output**:** top-k list S
Q : = empty priority queue
R : = 1
for $i = 0$ **to** $|V|$ **do**
    Update Q **with**$[(1, i), r[i]]$
end for
while Q is **not** empty **do**
    Pop an element $u = [(t, i), w]$ **with** the largest w from Q
    $S(u) = S(u) + (1 - a)* w$
    $R = R - (1 - a)* w$
for each $v = V(j)$ adjacent **to** $u = V(i)$ **do**
if$[(t + 1, j), x]$ is **in** Q **then**
        Update $[(t + 1, j), x]$ by $[(t + 1, j), x + a * W(E(u, v))]$
else
        Update Q **with**$[(t + 1, j), a * W(E(u, v))]$
end if
end for
if stop-condition$(R) <$ eps
for $i = 0$ **to** $|V|$ **do**
        Update S **with** the ordered $x > 0$ from $[(0, i), x]$
        break
end if
end while
return S

3. Path Selection Algorithm

In this section, we propose an efficient algorithm to find which nodes will be the top-k answers at an early stage. To show the benefit of our approach, we introduce a baseline method in advance. Initially, a random surfer starts from a node specified in the given vector $\vec{s}$ from the first state l_1. The total probability mass, which is specified as a *remainder*, is initialized as 1. The algorithm operates as a repetitive update of the node weights, which is the same as the random surfer's movements in the PPR model, in that the surfer moves to the next state or jumps to the special state l_0. The node weights are distributed and collected at each iteration. At the same time, the *remainder* value decreases with the amount of node weight assignments. The algorithm stops if the node weights in l_0 are

sufficiently stationary. To be specific, the algorithm stops when the *remainder* value is smaller than the truncation constant, ε. Because this is a probabilistic model, the initial probability mass is distributed to consecutive nodes with increasing time t. The sum of all node weights should be 1. This is called a brute-force algorithm (BFA), and is the baseline algorithm used.

The idea of this algorithm comes from the observance that if we know the stop condition earlier than confirmation of the full convergence of the node weights in l_0, we can obtain the top-k answers faster than the baseline BFA. The top-k query algorithm, called the PSA, maintains three values to check its termination based on the state of l_0, priority queue Q, top-k list S, and remainder R. Initially, a query node-set $\vec{s}$ is given. In the first state, l_1, each node obtains an initial node weight according to the probability distribution over vector $\vec{s}$, and the remainder r is initialized as 1. The nodes whose node weights are over zero are then queued up to Q. To process each state, the node whose weight is the largest is removed out from the queue. Some portions of its node weight are distributed to nodes in the next state, and some portion is accumulated into the zeroth state, l_0. The moving portions are determined by the edge connecting to other nodes and their edge weights. Specifically, the node weight of node $v_i^{(t)}$ is divided into the neighboring nodes in the next state, l_{t+1}. The portion $\alpha W(v_i^{(t)}) \times W\left(e_{ij}(v_i^{(t)}, v_j^{(t+1)})\right)$ is assigned to the next nodes, $v_j^{(t+1)}$. For the $(1-\alpha)$ part, the node weight $(1-\alpha)W(v_i^{(t)})$ is accumulated to the corresponding node, $v_i^{(0)}$. All neighboring nodes are updated and inserted into queue Q. The PSA repeats this process until a stop condition is satisfied. In the BFA, the stop condition is satisfied if there is no considerable change in the order of the top-k nodes in Q. In the PSA, we compare the k-th node with the $(k+1)$-th node in l_0 at each iteration. If the weight of the k-th node is larger than the upper bound of that of the $(k+1)$-th node, then we can terminate the algorithm immediately. The upper bound is easily estimated based on the remainder. The remainder is defined as the remaining probability mass from the given initial vector. This portion should decrease over time, and the jumps of the surfer make the portion smaller. We can use the fact that if the k-th node weight is larger than the sum of the remainder and the $(k+1)$-th node weight, further computations cannot change the order of the nodes. In this context, the epsilon value should be zero. For example, consider the graph with $|V| = 5$ with $\vec{s} = [0100]^{\mathrm{T}}$ and $\alpha = 0.75$ in Fig. 3. Node v_1 is given as a start node, and is assigned the probability mass 1 in the first state l_1. Initially, node v_1 in the first state is inserted into the queue in the form of [(node, state), weight]. The element [(v_1, 1), 1] is removed from Q, and its weight is then distributed over the neighboring nodes following the outgoing edges. Figure 3(b) shows a snapshot of the end of the first iteration. The nodes in the second state, v_2 and v_4, obtain 0.8×0.75 and 0.2×0.75, respectively. In addition, node v_1 in state l_0 obtains 0.25. The same amount will be subtracted from the remainder R to denote the remaining portion of the total probability to be distributed over all nodes. The priority queue Q is updated with [(v_1, 2), 0.6] and [(v_1, 4), 0.15] maintaining the descending order of the node weights. The top-k list S is updated with [v_1, 0.25]. As a subsequent step, Fig. 3(c) illustrates the updates and changes in Q, S, and R for the second iteration. From the queue, the element [(v_2, 2), 0.6] is removed. Following the outgoing edges of v_1, the portions 0.27, 0.135, and 0.045 are distributed to the nodes v_0, v_1, and v_3 in state l_3, respectively. In the priority queue, the order is changed owing to the incoming node v_0 with a weight of 0.27. The remainder will decrease from 0.75 to 0.6. In this setting, we only need to observe the remainder constant and k-th element in the top-k list. Furthermore, we can easily extend the algorithm using tighter stop conditions or path finding algorithms.

IV. Evaluation

In this section, we present the results of experiments assessing the efficiency and accuracy of the proposed algorithms. We compare the PSA with two competitive algorithms, a basic push algorithm (BPA) [13] and a KDA [12], as described earlier.

1. Experimental Settings

We used two real-world datasets, the Astrophysics dataset and LiveJournal dataset provided by the Stanford Large Network Dataset Collection (snap.stanford.edu/data). An Astrophysics graph was collected from arXiv and it covers papers submitted to the astrophysics category. This graph consists of 18,772 nodes and 198,110 edges. The LiveJournal dataset is a friend network extracted from an online community maintaining journals and group blogs. This graph consists of 3,137,571 million nodes and 29,552,850 edges. We randomly selected 1,000 nodes, each of which is used as a test query. Based on the linearity theorem [4], multiple query nodes can be easily adopted. For each algorithm, we conducted 1,000 evaluations for each query, and obtained the average performance values. Each algorithm was executed to retrieve the top-k nodes according to the PPR scores. We used the execution time as the metric of efficiency. To measure the accuracy, we used the fraction of answer nodes among the top-k answers from each method that match those of the BFA that produce the exact answers. We varied constant k as 10,

20, 30, 40, and 50, and used the restart probability $1 - \alpha = 0.05$, as in previous works [5], [12]. For BPA, 20%, 25%, and 30% of the nodes were randomly selected as hub nodes, and truncation constant ε was fixed at 10^{-5}. For accuracy assessments of BFA and PSA, we varied ε as 0.01, 0.05, 10^{-3}, 5×10^{-3}, 10^{-4}, 5×10^{-4}, 10^{-5} and 5×10^{-5}. All of our experiments were conducted on a 3.0 GHz Intel Xeon X5472 machine with 32 GB of memory running Ubuntu 10. Each method was implemented using C++. A binomial heap [27] is used as the priority queue in the PSA and BFA.

2. Results

Table 1 shows the pre-computation times of the different algorithms. The BPA needs to select hub nodes to estimate the upper bounding proximities. A recent approach, the KDA, proposes an efficient decomposition method to compute the inverse matrices. However, it still needs several hours of pre-computation time. It is important to note that the BPA is an approximate algorithm and the KDA is an exact algorithm. The KDA, which derives an inverse matrix at the pre-computation time, is infeasible for a large graph. Note that the PSA does not require any pre-computations compared to the other algorithms. This benefit is critical for online algorithms with large graphs.

Figures 4 and 5 show the performance results for the Astrophysics dataset, which is a smaller dataset than the LiveJournal dataset. Figure 4 shows that the PSA outperforms the BPA with respect to the execution time. If we exclude the pre-computation time from the BPA, which computes the hub nodes, the PSA operates much more quickly than the BPA because it requires additional time to compute the PPR scores for non-hub nodes. Moreover, the BPA processes all of the paths evenly, whereas the PSA processes the important paths in a greedy fashion. Although the KDA is the fastest algorithm for online use, the PSA is comparable. However, if the pre-computation time is considered as the total computation time, the PSA is the fastest. Figure 5 shows the accuracy of the approximate algorithm, BPA, compared to the exact algorithm, KDA. Although the PSA produces approximate results, it has a slight loss of accuracy for an increasing k compared to the BPA. The KDA consistently outperforms the PSA in terms of efficiency and accuracy after pre-computing the matrix inversion. The exact algorithm, KDA, guarantees exact answers for the PPR, but requires $O(n^2)$ time and space for computing all node proximities, and is bounded by $O(n + m)$ for the top-k answers. In some cases with a large ratio of the number of non-zero elements in the matrix, the efficiency of the KDA worsens. In addition, the PSA consistently performs well without depending on the element structures.

Figures 6 and 7 show the results of the same evaluations described previously on the larger LiveJournal dataset. The pre-computation of the KDA is infeasible. Therefore, it cannot be applied to large graphs. The PSA consistently outperforms the BPA in both efficiency and accuracy.

Table 1. Pre-computation time (s).

	LiveJournal	Astrophysics
BFA	0	0
BPA20	12,128	2,506
BPA25	28,300	3,165
BPA30	40,899	13,487
KDA	N/A	76,493
PSA	0	0

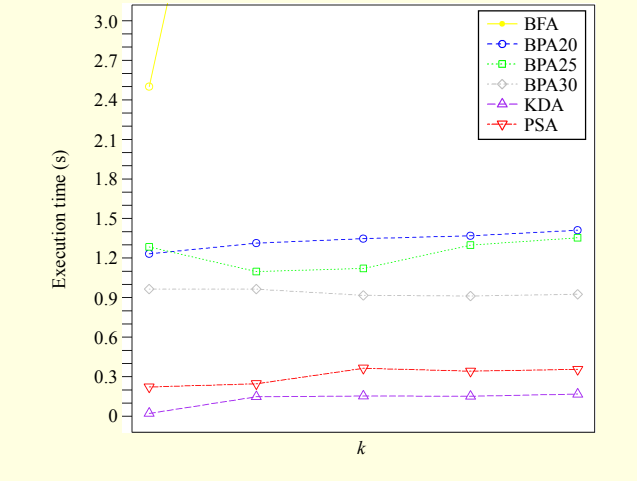

Fig. 4. Efficiency (Astrophysics).

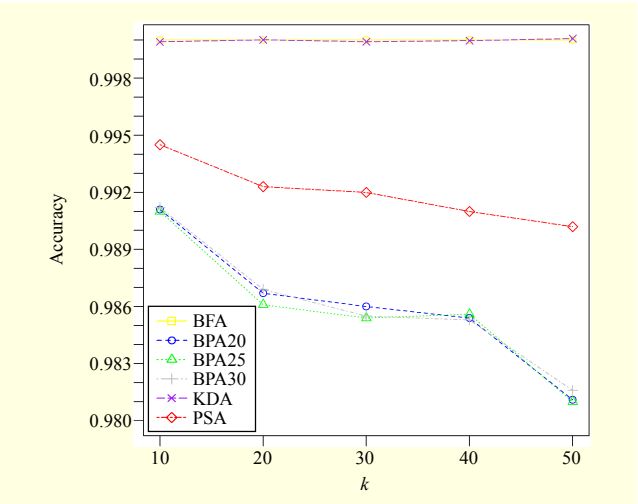

Fig. 5. Accuracy (Astrophysics).

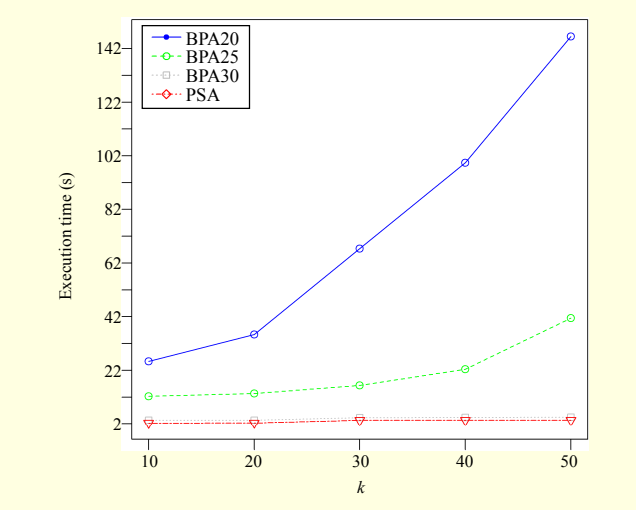

Fig. 6. Efficiency (LiveJournal).

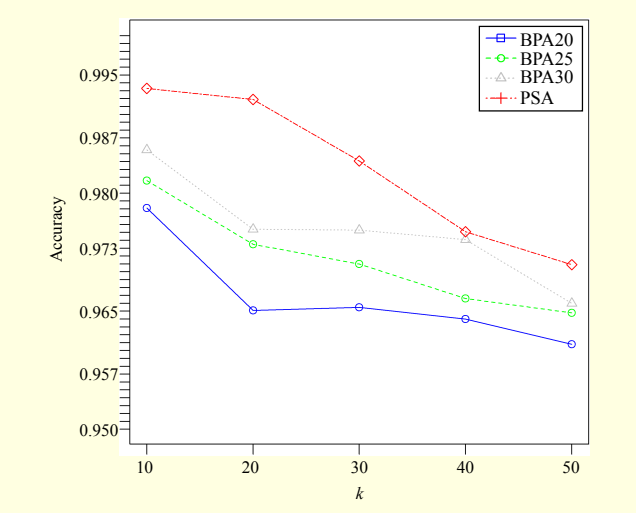

Fig. 7. Accuracy (LiveJournal).

Figure 8 shows that the accuracy of the PSA improves with smaller truncation constants. The PSA takes more paths into consideration as ε is smaller, but the paths have little influence on the top-k node determination. Hence, it is important to choose an appropriate ε because the performance relies on it. If ε is too small, the PSA will derive a small accuracy gain but consume a large amount of computational time. On the other hand, if ε is too large, the PSA will operate very quickly with a huge loss of accuracy.

V. Conclusion

In this paper, we presented a novel method to find the top-k answers in a node proximity search based on the PPR model. To compare the random paths with different time stages to promising nodes, we extended a given graph to a novel

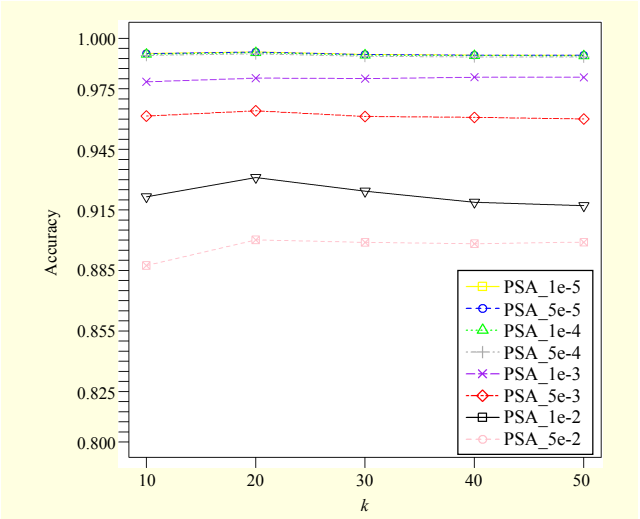

Fig. 8. Truncation of PSA.

structure, a DSTG. Based on the notions of time sequences and weight distribution models, the top-k answers were efficiently computed without pre-computations. We proposed an algorithm, the PSA, to maintain a promising node list and test the upper bounds for an early determination. We proved that our answers determined through a DSTG guarantee equivalence with the results of the PPR. A performance study using two real datasets clearly shows that our method is practical and accurate. As future work, we plan to parallelize the PSA to handle data on the *Spark* framework. Moreover, we are considering applying our method to other random walk-based measures [6], [28].

References

[1] B.-W. On et al., "Comparative Study of Name Disambiguation Problem using a Scalable Blocking-Based Framework," *ACM/IEEE-CS Joint Conf. Digi. Libraries*, NY, USA, June 7–11, 2005, pp. 344–353.

[2] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating Web Spam with Trustrank," *Int. Conf. Very Large Databases*, Toronto, Canada, Aug. 29–Sept. 3, 2004, pp. 576–587.

[3] H. Yanagimoto and M. Yoshioka, "Relationship Strength Estimation for Social Media Using Folksonomy and Network Analysis," *IEEE Int. Conf. Fuzzy Syst.*, Brisbane, Austrailia, June 10–15, 2012, pp. 1–8.

[4] G. Jeh and J. Widom, "Scaling Personalized Web Search," *Int. Conf. World Wide Web*, Budapest, Hungary, May 20–24, 2003, pp. 271–279.

[5] H. Tong, C. Faloutsos, and J.Y. Pan, "Fast Random Walk with Restart and its Applications," *IEEE Int. Conf. Data Mining*, Hong Kong, Dec. 18–22, 2006, pp. 613–622.

[6] G. Jeh and J. Widom, "SimRank: A Measure of Structural-Context Similarity," *ACM Int. Conf. Knowl. Discovery Data Mining*, Alberta, Canada, July 23–26, 2002, pp. 538–543.

[7] X. Ye and T. Sakurai, "Robust Similarity Measure for Spectral Clustering Based on Shared Neighbors," *ETRI J.*, vol. 38, no. 3, June 2016. pp. 540–550.

[8] S. Govindaraj and K. Gopalakrishnan, "Intensified Sentiment Analysis of Customer Product Reviews Using Acoustic and Textual Features," *ETRI J.*, vol. 38, no. 3, June 2016, pp. 494–501.

[9] H. Jeon and S. Lee, "Language Model Adaptation Based on Topic Probability of Latent Dirichlet Allocation," *ETRI J.*, vol. 38, no. 3, June 2016, pp. 487–493.

[10] B. Bahmani, A. Chowdhury, and A. Goel, "Fast Incremental and Personalized Pagerank," *Int. Conf. Very Large Databases*, Singapore, Sept. 13–17, 2010, pp. 173–184.

[11] D. Fogaras et al., "Towards Scaling Fully Personalized Pagerank: Algorithms, Lower Bounds, and Experiements," *Internet Math.*, vol. 2, no. 3, 2005, pp. 333–358

[12] Y. Fujiwara et al., "Efficient Personalized Pagerank with Accuracy Assurance," *ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Beijing, China, Aug. 12–16, 2012, pp. 15–23.

[13] M. Gupta, A. Pathak, and S. Chakrabarti, "Fast Algorithms for Top-*k* Personalized Pagerank Queries," *Int. Conf. World Wide Web*, Beijing, China, Apr. 21–25, 2008, pp. 1225–1226.

[14] F. Zhu et al., "Incremental and Accuracy-Aware Personalized Pagerank through Scheduled Approximation," *Proc. VLDB Endowment*, vol. 6, no. 6, Apr. 2013, pp. 481–492.

[15] B. Bahmani, K. Chakrabarti, and D. Xin, "Fast Personalized Pagerank on Mapreduce," *ACM Int. Conf. Manag. Data*, Athens, Greece, June 12–16, 2011, pp. 973–984.

[16] T.H. Haveliwala, "Topic-Sensitive Pagerank," *Int. Conf. World Wide Web*, Honolulu, HI, USA, May 7–11, 2002, pp. 517–526.

[17] P. Berkhin, "Bookmark-Coloring Algorithm for Personalized Pagerank Computing," *Internet Math.*, vol. 3, no. 1, 2006, pp. 41–61.

[18] K. Avrachenkov et al., "Quick Detection of Top-k Personalized Pagerank Lists," *Int. Workshop Algorithms Models Web-Graph*, Atlanta, GA, USA, May 27–29, 2011, pp. 50–61.

[19] S. Brin and L. Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine," *Int. Conf. World Wide Web*, Brisbane, Australia, Apr.14–18, 1998, pp. 107–117.

[20] G. Strang. "*Introduction to Linear Algebra*," Welleselly, MA, USA: Wellesley-Cambridge Press, 2009.

[21] J. Huang, X. Zhang, Y. Zhang, X. Zou, and L. Zeng, "Speech Denoising via Low-Rank and Sparse Matrix Decomposition," *ETRI J.*, vol. 36, no. 1, Feb. 2014, pp. 167–170.

[22] S.D. Kamvar et al., "Extrapolation Methods for Accelerating Pagerank Computations," *Int. Conf. World Wide Web*, Budapest, Hungary, May 20–24, 2003, pp. 261–270.

[23] R. Andersenet al., "Local Computation of Pagerank Contributions," *Int. Conf. Algorithms Models Web-Graph*, San Diego, CA, USA, Dec. 11–12, pp. 150–165.

[24] Y. Fujiwara et al., "Fast and Exact Top-*k* Search for Random Walk with Restart," *Proc. VLDB Endowment*, vol. 5, no. 3, 2012, pp. 442–453.

[25] Y. Wu et al., "Fast and Unified Local Search for Random Walk Based k-Nearest-Neighbor Query in Large Graphs," *ACM Int. Conf. Manag. Data*, Snowbird, UT, USA, June 22–27, 2014, pp. 1139–1150.

[26] A.W.Yu, N. Mamoulis, and H. Su, "Reverse Top-*k* Search Using Random Walk with Restart," *Proc. VLDB Endowment*, vol. 7, no. 5, 2014, pp. 401–412.

[27] J. Vuillemin, "A Data Structure for Manipulating Priority Queues," *Commun. ACM*, vol. 21, no.4, Apr. 1978, pp. 309–315.

[28] Y. Sun et al., "PathSim: Meta Path-Based Top-*k* Similarity Search in Heterogeneous Information Networks," *Int. Conf. Very Large Databases*, Seattle, WA, USA, Aug. 29–Sept. 3, 2011, pp. 992–1003.

Jaehui Park received his BS degree in computer science from Korea Advanced Institute Science and Technology, Daejeon, Rep. of Korea in 2005, and his PhD in computer science and engineering from Seoul National University, Rep. of Korea. He is currently a senior research engineer at ETRI, Daejeon, Rep. of Korea. His research interests include keyword searches in relational databases, information retrieval, and text mining.

Sang-Goo Lee received his BS degree in computer science and statistics from Seoul National University, Rep. of Korea in 1985, and his MS and PhD degrees at the Department of Computer Science from Northwestern University, Evanston, IL, USA, in 1987 and 1990, respectively. He is currently a professor of computer science at Seoul National University. His research interests are in semantic technology, context-aware personalization, e-catalogs, and database design.