gIceberg: Towards Iceberg Analysis in Large Graphs

Nan Li[†], Ziyu Guan[†], Lijie Ren[†], Jian Wu[§], Jiawei Han[‡], Xifeng Yan[†]

[†]*Department of Computer Science, University of California at Santa Barbara*
Santa Barbara, CA 93106, USA
[†]{nanli, ziyuguan, lijie, xyan}@cs.ucsb.edu
[§]*College of Computer Science & Technology, Zhejiang University*
Hangzhou, Zhejiang 310027, China
[§]wujian2000@zju.edu.cn
[‡]*Department of Computer Science, University of Illinois at Urbana-Champaign*
Urbana, IL 61801, USA
[‡]hanj@cs.uiuc.edu

***Abstract*—Traditional multi-dimensional data analysis techniques such as iceberg cube cannot be directly applied to graphs for finding interesting or anomalous vertices due to the lack of dimensionality in graphs. In this paper, we introduce the concept of *graph icebergs* that refer to vertices for which the concentration (aggregation) of an attribute in their vicinities is abnormally high. Intuitively, these vertices shall be "close" to the attribute of interest in the graph space. Based on this intuition, we propose a novel framework, called gIceberg, which performs aggregation using random walks, rather than traditional SUM and AVG aggregate functions. This proposed framework scores vertices by their different levels of interestingness and finds important vertices that meet a user-specified threshold. To improve scalability, two aggregation strategies, forward and backward aggregation, are proposed with corresponding optimization techniques and bounds. Experiments on both real-world and synthetic large graphs demonstrate that gIceberg is effective and scalable.**

I. Introduction

The ubiquity of large-scale graphs has motivated research in graph mining and analysis, such as frequent graph pattern mining [1], graph summarization/compression [2], and graph anomaly detection [3]. An important feature of real-world graphs is that they often contain attributes on their vertices. For instance, in an academic collaboration network, a vertex is an author and the vertex attributes can be their research topics. In a customer social network, the vertex attributes can be the products the customers purchased. Various studies have been dedicated to mining attributed graphs [4], [5], [6].

Given a large vertex attributed graph, how can we find interesting vertices or anomalies? Certainly, the interestingness criteria vary among different applications [7], [3], [8]. In this paper, we introduce a generic concept of *graph icebergs* that refer to vertices for which the concentration (aggregation) of an attribute in their vicinities is abnormally high. The name, "iceberg", is borrowed from the concept of iceberg queries proposed in [9]. When querying traditional relational databases, many applications entail computing aggregate functions over an attribute (or a set of attributes) to find aggregate values above some specified threshold. Such queries are called *iceberg queries*, because the number of above-threshold results is often small (the tip of an iceberg), compared to the large amount of input data [9]. Analogously, an aggregate function, such as the percentage of neighboring vertices containing the attribute, can be applied to each vertex in the graph, to assess the concentration of a certain attribute within the vertex's vicinity. An aggregate score is computed for each vertex's vicinity. Graph iceberg vertices are retrieved as those whose aggregate score is above a given threshold.

Applications of graph iceberg mining abound, including target marketing, recommendation systems, social influence analysis, intrusion detection, and so on. In a social network, if many of John Doe's friends bought an iPhone but he has not, he would be a good target for iPhone promotion, since he could be influenced by his friends. In a geographic network, we can find sub-networks where crimes occur more often than the rest of the network. The detection of such sub-networks could help law enforcement officers better allocate their resources. In addition, if the detected iceberg vertices form sparse subgraphs, social influence analysis can be applied, since sparse edge connections among iceberg vertices often indicate social influence, rather than homophily [10].

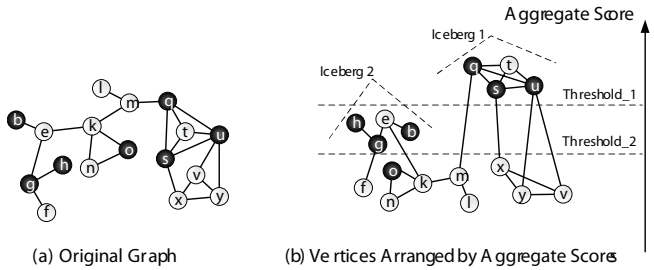


Fig. 1. Graph Iceberg

Figure 1(a) shows a vertex-attributed graph, where black vertices are those containing the attribute of interest, which could be a product purchase, a network attack, a reported crime, etc. An aggregate score is computed for each vertex, indicating the concentration level of the attribute in its vicinity. Figure 1(b) shows that the vertices can be rearranged according to their aggregate scores. Vertices with higher scores are positioned higher. By inserting cutting thresholds with different values, we can retrieve different sets of iceberg vertices.

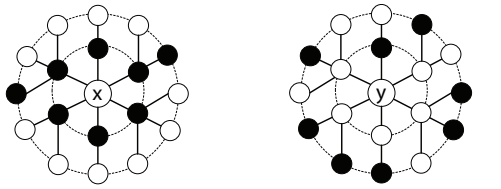


Fig. 2. PPV Aggregation vs. Other Aggregation Measures

These retrieved icebergs can be further processed to form connected subgraphs. Those subgraphs contain only vertices whose local neighborhoods exhibit high concentration of an attribute of interest. Such analysis will be very convenient for users to explore large graphs since they can focus on just a few, important vertices and by varying the thresholds and the attributes of interest, they can change their focus. Note that this differs from dense subgraph mining and clustering, since connected subgraphs formed by iceberg vertices do not necessarily have high edge connectivity.

Now the question is what kind of aggregate functions one shall use to find iceberg vertices? There are many possible measures to describe a vertex's local vicinity. Yan et al. proposed two aggregate functions over a vertex's h-hop neighborhood, SUM and AVG [5]. In our scenario, for a vertex v, SUM and AVG compute the number and percentage of black vertices in v's h-hop neighborhood, respectively. However, we argue that SUM and AVG fail to effectively evaluate how close a vertex is to an attribute. Figure 2 shows the 2-hop neighborhoods of vertices x and y. Both x and y have 18 neighbors that are within 2-hop distance away. For both x and y, 8 out of the 18 2-hop neighbors are black; namely, 8 out of 18 2-hop neighbors contain the attribute of interest. Therefore, both SUM and AVG will return the same value on x and y. However, x is clearly "closer" to the attribute than y, because all of x's adjacent neighbors are black, whereas most of y's black 2-hop neighbors are located on the second hop. Thus we need a different aggregate function to better evaluate the proximity between a vertex and an attribute.

In this paper, we use the random walk with restart model [11] to weigh the vertices in one vertex's vicinity. A random walk is started from a vertex v; at each step the walker has a chance to be reset to v. This results in a stationary probability distribution over all the vertices, denoted by the personalized PageRank vector (PPV) of v [11]. The probability of reaching another vertex u reflects how close v is to u with respect to the graph structure. The concentration of an attribute q in v's local neighborhood is then defined as the aggregation of the entries in v's PPV corresponding to those vertices containing the attribute q, namely the total probability to reach a node containing q from v. This definition reflects the *affinity*, or *proximity*, between vertex v and attribute q in the graph. In Figure 2, by aggregating over x and y's PPVs, we can capture the fact that x is in a position more abnormal than y, since x has a higher aggregate score than y.

As an alternative, we can use the shortest distance from vertex v to attribute q to measure v's affinity to q. However, shortest distance does not reflect the overall distribution of q in v's local neighborhood. It is possible that the shortest distance is small, but there are only few occurrences of q in v's local neighborhood. In the customer network example, if John has a close friend who purchased an iPhone, and this friend is the only person John knows that did, John might not be a promising candidate for iPhone promotion.

With such PPV-based definition of graph icebergs, we design a scalable framework, called glceberg, to compute the proposed aggregate measure. Vertices whose measure is above a threshold are retrieved as graph iceberg vertices. These iceberg vertices can be further processed (e.g. graph clustering) to discover *graph iceberg regions*. Section VI discusses an interesting clustering property of iceberg vertices, which facilitates the discovery of those regions. We will also show in our experiments that glceberg discovers interesting author groups from the DBLP network.

Our Contributions. (1) A novel concept, graph iceberg, is introduced. (2) glceberg finds iceberg vertices in a scalable manner, which can be leveraged to further discover iceberg regions. (3) Two aggregation methods are designed to quickly identify iceberg vertices, which hold their own stand-alone technical values. (4) Experiments on real-world and synthetic graphs show the effectiveness and scalability of glceberg.

II. Preliminaries

Previous studies [12] showed that personalized PageRank (PPR) measures the *proximity* of two vertices. If the aggregation is done on a PPV with respect to an attribute, the aggregate score naturally reflects the concentration of that attribute within a vertex's vicinity. Given an undirected graph $G = (V, E, \mathbb{L})$, where V is the set of vertices, E is the set of edges, and $\mathbb{L} = \{q_1, \ldots, q_l\}$ is the set of attributes, let $L(v) \subseteq \mathbb{L}$ be the set of attributes v contains. This paper focuses on boolean attributes, which means for an attribute q, a vertex either contains it or not. A *query* is an attribute in $\mathbb{L}$. Our framework can be extended to queries with multiple attributes, scalar attributes and edge-weighted graphs.

Let $\boldsymbol{M}$ be the transition matrix. $\boldsymbol{M}_{ij} = 1/d_{v_j}$ if there is an edge between vertices v_i and v_j; and 0 otherwise. d_{v_j} is the vertex degree of v_j. c is the restart probability in the random walk model. A preference vector $\boldsymbol{s}$, where $|\boldsymbol{s}|_1 = 1$, encodes the amount of preference for each vertex. PageRank vector $\boldsymbol{p}$ is defined as the solution of Equation (1) [11]:

$$\boldsymbol{p} = (1 - c)\boldsymbol{M}\boldsymbol{p} + c\boldsymbol{s}. \quad (1)$$

If $\boldsymbol{s}$ is uniform over all vertices, $\boldsymbol{p}$ is the *global PageRank vector*. For nonuniform $\boldsymbol{s}$, $\boldsymbol{p}$ is the *personalized PageRank vector* of $\boldsymbol{s}$. When $\boldsymbol{s} = \mathbf{1}_v$, where $\mathbf{1}_v$ is the unit vector with value 1 at entry v and 0 elsewhere, $\boldsymbol{p}$ is the personalized PageRank vector of vertex v, also denoted as $\boldsymbol{p}_v$. The xth entry in $\boldsymbol{p}_v$, $\boldsymbol{p}_v(x)$, reflects the importance of x in the view of v. In this paper, we typeset vectors in boldface (e.g., $\boldsymbol{p}_v$) and use parentheses to denote an entry in the vector (e.g., $\boldsymbol{p}_v(x)$).

Definition 1 (Black Vertex): For an attribute q, a vertex that contains the attribute q is called a *black vertex*.

We propose to aggregate over the PPV to define the closeness of a vertex to an attribute q.

Definition 2 (q-Score): For an attribute q, the q-score of v is defined as the aggregation over v's PPV:

$$\mathcal{P}_q(v) = \Sigma_{x|x\in V, q\in L(x)} \boldsymbol{p}_v(x), \tag{2}$$

where $\boldsymbol{p}_v$ is the PPV of v.

The q-score is the sum of the black vertices' entries in a vertex's PPV. Calculating q-scores for a query is called *personalized aggregation*. A vertex with a high q-score has a large number of black vertices within its local neighborhood. Intuitively, q-score measures the probability for a vertex v to reach a black vertex, in a random walk starting from v after the walk converges.

Definition 3 (Iceberg Vertex): For a vertex v, if its q-score is above a certain threshold, v is called an iceberg vertex; otherwise it is called a non-iceberg vertex.

Problem 1 (Graph Iceberg Problem): For an undirected vertex-attributed graph $G = (V, E, \mathbb{L})$ and a query attribute q, the graph iceberg problem finds all the iceberg vertices given a user-specified threshold θ.

The *power method* computes the PageRank vectors iteratively as in Equation (1) until convergence [13], which is expensive for large graphs. Random walks were used to approximate both global and personalized PageRank [14], [15], [16]. The PPV of vertex v, $\boldsymbol{p}_v$, can be approximated using a series of random walks starting from v, each of which continues until its first restart. The length of each walk follows a geometric distribution. [16] discovered the following theory: Consider a random walk starting from v and taking L steps, where L follows a geometric distribution $\Pr[L = i] = c(1-c)^i, i = \{0, 1, 2, \ldots\}$ with c as the restart probability, then the PPR of vertex x in the view of v is the probability that the random walk ends at x. Combining this with the *Hoeffding Inequality* [17], we establish probabilistic bounds on approximating PPRs using random walks.

Theorem 1 (PPV Approximation): Suppose R random walks are used starting from vertex v to approximate v's PPV, $\boldsymbol{p}_v$. Let $\tilde{\boldsymbol{p}}_v(x)$ be the percentage of those R walks ending at x, then we have $\Pr[\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x) \geq \epsilon] \leq \exp\{-2R\epsilon^2\}$ and $\Pr[|\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$, for any $\epsilon > 0$.

The proof is in the appendix. Theorem 1 states that if enough random walks are used, the error between approximate and actual PPR is limited probabilistically. For example, if $\epsilon = 0.05$ and $R = 500$, $\Pr[\boldsymbol{p}_v(x) - \epsilon \leq \tilde{\boldsymbol{p}}_v(x) \leq \boldsymbol{p}_v(x) + \epsilon] \geq 83.58\%$. We use such random walks in glceberg to approximate PPVs on large graphs.

III. Framework Overview

We propose the glceberg framework to discover graph iceberg vertices. It takes two steps: (1) user specifies a query attribute q and a q-score cut-off threshold θ; and (2) glceberg identifies vertices whose q-score is above θ. Vertices whose q-score is below θ are pruned. For better efficiency, random walks are used to approximate PPVs and the aggregation is done on approximate PPVs. Algorithm 1 gives an overview of glceberg. Due to the error introduced by approximation, glceberg returns vertices whose approximate q-score is above the threshold minus an error tolerance, $\theta - \epsilon$. The accuracy of such process will be analyzed later.

Algorithm 1: The glceberg Framework

Input: G, query q, threshold θ, approximate error ϵ
Output: Graph iceberg vertices
1 Apply random walks to get approximate PPVs;
2 Perform aggregation over approximate PPVs to compute approximate q-scores;
3 Return vertices whose approximate q-score is above $\theta - \epsilon$;

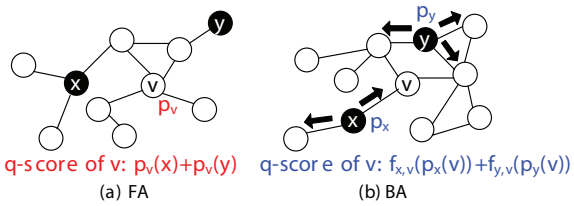


Fig. 3. Forward & Backward Aggregation

One way to properly set the threshold θ is to consider θ as the *significance level* of q-scores. Namely, θ can be chosen via assessing the distribution of vertex q-scores in random cases. We can randomly permute the attribute q among the vertices in the graph and calculate the empirical distribution of vertex q-scores; then we choose a point in the distribution to be θ so that only a small percentage (e.g., 5%) of vertices in the distribution have q-scores higher than θ.

The core of glceberg is the aggregation over PPVs. Two efficient aggregation schemes, *forward aggregation* (FA) and *backward aggregation* (BA), are proposed with respective optimization techniques. FA computes the q-score by adding the PPR scores of all the black vertices for the current vertex; BA starts from the black vertices and back-propagates their PPR scores to other vertices. In Figure 3(a), v's q-score is the sum of the PPR scores of x and y in v's PPV: $\boldsymbol{p}_v(x) + \boldsymbol{p}_v(y)$. In Figure 3(b), the bold black arrows indicate the direction of PageRank aggregation, which starts from black vertices, and propagates backward to other vertices. For black vertex x, its contribution to v's q-score can be written as a function of v's PPR with respect to x: $f_{x,v}(\boldsymbol{p}_x(v))$. v's q-score is the sum of the contributions of all the black vertices. Approximate aggregation will also be introduced and analyzed later.

IV. Forward Aggregation

Basic FA uses random walks to approximate the PPVs. Aggregation is subsequently conducted over the approximate PPVs. We then propose optimization techniques for FA, including Pivot vertex-based FA (PFA) that is designed to avoid a linear scan of all vertices. PFA incorporates various q-score bounds for efficient vertex pruning.

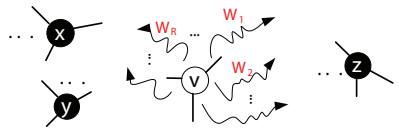


Fig. 4. Forward Aggregation Approximation

A. Forward Aggregation Approximation

Applying FA on approximate PPVs generated by random walks is called *FA approximation*, as shown in Figure 4. For each vertex v, R random walks, $\{W_1, \ldots, W_R\}$, are conducted starting from v, to approximate v's PPV. Each walk continues until its first restart. Once the approximate PPV, $\tilde{\boldsymbol{p}}_v$, is derived, the approximate q-score of v is the sum of the entries in $\tilde{\boldsymbol{p}}_v$ corresponding to the black vertices. We analyze the accuracy of such approximate aggregation by using the Hoeffding Inequality as in Theorem 2.

Theorem 2 (FA Approximation): Suppose we perform R random walks from vertex v to compute v's approximate PPV, $\tilde{\boldsymbol{p}}_v$. Let $\tilde{\mathcal{P}}_q(v)$ be the approximated q-score of v. We have $\Pr[\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v) \geq \epsilon] \leq \exp\{-2R\epsilon^2\}$ and $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$, for any $\epsilon > 0$.

The proof is in the appendix. Now we analyze how well FA retrieves *real* iceberg vertices. As in Algorithm 1, FA retrieves all vertices whose approximate q-score is above $\theta - \epsilon$. We use *recall* to measure the accuracy of such retrieval. For certain θ and ϵ, recall is computed as $\big|\{v|\mathcal{P}_q(v) \geq \theta, \tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\big| / \big|\{v|\mathcal{P}_q(v) \geq \theta\}\big|$. Recall is the percentage of real iceberg vertices that are retrieved by the approximate aggregation.

Corollary 1 (FA Recall): Given a q-score cut-off threshold θ, for vertex v such that $\mathcal{P}_q(v) \geq \theta$, we have $\Pr[\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$, where $\epsilon > 0$ and $\tilde{\mathcal{P}}_q(v)$ is v's q-score using FA approximation.

Proof: The proof follows from Theorem 2. ■

Therefore, if we use $\theta - \epsilon$ as the threshold on approximate q-scores to retrieve iceberg vertices, Corollary 1 says that we can derive a theoretical lower bound for the expected recall, i.e., $\Pr[\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon]$, where v is an iceberg vertex.

B. Improving Forward Aggregation

Although FA simulates random walks to estimate PPVs, it still calculates the PPV for each vertex. In this section, we propose pruning techniques to avoid computing all the approximate PPVs. We first adapt the decomposition property of PPVs to the case of q-scores (Theorem 3). This property means that we can bound the q-scores of v's neighbors if we know v's (Corollary 2). We then further develop a better bound for the 2-hop neighbors by exploiting the common neighbors of two vertices (Theorem 4). Finally, we establish "pivot-client" relations between vertices and use q-scores of the pivot vertices to prune client vertices.

1) Aggregation Decomposition: We first introduce the q-score decomposition property. Previous studies proposed PPV *Decomposition Theorem* [18], which expresses the PPV of a vertex in terms of those of its adjacent neighbors.

$$\boldsymbol{p}_v = \frac{1-c}{|N_1(v)|}\Sigma_{x \in N_1(v)}\boldsymbol{p}_x + c\mathbf{1}_v, \tag{3}$$

where $N_1(v)$ is the set of 1-hop neighbors of v, c is the restart probability, and $\mathbf{1}_v$ is the unit vector with value 1 at entry v and 0 elsewhere. Let $d_v = |N_1(v)|$. We find that similar decomposition can be applied on q-scores.

Theorem 3 (q-Score Decomposition): Given a query attribute q, the q-score of a vertex $v \in V$, $\mathcal{P}_q(v)$, can be expressed via those of its neighbors as follows,

$$\mathcal{P}_q(v) = \frac{1-c}{d_v}\Sigma_{x \in N_1(v)}\mathcal{P}_q(x) + c1_{q \in L(v)}, \tag{4}$$

where $1_{q \in L(v)}$ is an indicator function: $1_{q \in L(v)} = 1$ if q is an attribute of vertex v, and $1_{q \in L(v)} = 0$ otherwise.

Proof: According to Definition 2 and Equation (3):

$$\begin{aligned}
\mathcal{P}_q(v) &= \Sigma_{y|y \in V, q \in L(y)}\boldsymbol{p}_v(y) \\
&= \Sigma_{y|y \in V, q \in L(y)}\Big(\frac{1-c}{|N_1(v)|}\Sigma_{x \in N_1(v)}\boldsymbol{p}_x(y) + c\mathbf{1}_v(y)\Big) \\
&= \frac{1-c}{d_v}\Sigma_{x \in N_1(v)}\Sigma_{y|y \in V, q \in L(y)}\boldsymbol{p}_x(y) \\
&\quad + c\Sigma_{y|y \in V, q \in L(y)}\mathbf{1}_v(y) \\
&= \frac{1-c}{d_v}\Sigma_{x \in N_1(v)}\mathcal{P}_q(x) + c1_{q \in L(v)}.
\end{aligned}$$

Therefore, Theorem 3 is proven. ■

2) q-Score Bounds: Theorem 3 expresses the q-score of a vertex in terms of those of its adjacent neighbors. Therefore, if the q-score of a vertex is known, we can derive an upper bound on the q-scores of this vertex's neighbors. If such an upper bound is smaller than threshold θ, we can prune those neighbors without actually computing their q-scores.

Corollary 2 (Neighbor q-Score Bound): Given a query attribute q, for any vertex $v \in V$, its q-score, $\mathcal{P}_q(v)$, and the q-score of any of v's neighbor x, $\mathcal{P}_q(x)$, satisfy $\mathcal{P}_q(x) \leq \frac{d_v}{1-c}(\mathcal{P}_q(v) - c1_{q \in L(v)})$.

Proof: The proof follows from Theorem 3. ■

The bound in Corollary 2 could be loose since $\frac{d_v}{1-c}$ is always greater than 1. For vertices with moderate degrees, the bound can easily exceed 1, making it a trivial bound. Next we propose a better bound for the 2-hop neighborhoods. We define the pivot-client (PC) relation between two vertices having similar neighborhoods. If two vertices u and v have similar 1-hop neighborhoods, namely $N_1(u)$ and $N_1(v)$ overlap, we can use the q-score of u to bound that of v, and vice versa (Theorem 4).

Theorem 4 (PC q-Score Bound): Suppose we have two vertices u and v. $N_1(u) \cap N_1(v)$ is not empty. Let $\sigma_u = |N_1(u) \cap N_1(v)|/|N_1(u)|$ and $\sigma_v = |N_1(u) \cap N_1(v)|/|N_1(v)|$. Then the q-scores of u and v satisfy: $\mathcal{P}_q(v) \leq \mathcal{P}_q(u)d_u/d_v + c(1_{q \in L(v)} - 1_{q \in L(u)}d_u/d_v) + (1-c)(1-\sigma_v)$.

Proof: Let $C = N_1(u) \cap N_1(v)$ denote the set of common neighbors shared between u and v. Therefore, we have $\sigma_u = |C|/|N_1(u)|$ and $\sigma_v = |C|/|N_1(v)|$. According to Theorem 3:

$$\mathcal{P}_q(u) = \frac{1-c}{d_u}\big(\Sigma_{x \in C}\mathcal{P}_q(x) + \Sigma_{x \in N_1(u) \setminus C}\mathcal{P}_q(x)\big) + c1_{q \in L(u)},$$

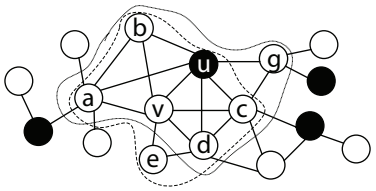


Fig. 5. Pivot-Client Relation

Likewise, we have

$$\mathcal{P}_q(v) = \frac{1-c}{d_v}\big(\Sigma_{x\in C}\mathcal{P}_q(x) + \Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x)\big) + c1_{q\in L(v)}.$$

Combining the above two equations, we have:

$$\begin{aligned}\mathcal{P}_q(v) =& \frac{1-c}{d_v}\big(\frac{d_u}{1-c}(\mathcal{P}_q(u) - c1_{q\in L(u)}) \\ & - \Sigma_{x\in N_1(u)\setminus C}\mathcal{P}_q(x) + \Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x)\big) + c1_{q\in L(v)} \\ =& \frac{d_u}{d_v}\mathcal{P}_q(u) + c\big(1_{q\in L(v)} - \frac{d_u}{d_v}1_{q\in L(u)}\big) \\ & + \frac{1-c}{d_v}\big(\Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x) - \Sigma_{x\in N_1(u)\setminus C}\mathcal{P}_q(x)\big) \\ \leq& \frac{d_u}{d_v}\mathcal{P}_q(u) + c\big(1_{q\in L(v)} - \frac{d_u}{d_v}1_{q\in L(u)}\big) \\ & + \frac{1-c}{d_v}\Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x). \end{aligned} \quad (5)$$

Since all entries of a PPV add up to 1, the aggregate value for any vertex v, $\mathcal{P}_q(v) \leq 1$. Therefore, we have:

$$\begin{aligned}\frac{1-c}{d_v}\Sigma_{x\in N_1(v)\setminus C}\mathcal{P}_q(x) \leq& \frac{1-c}{d_v}(|N_1(v)| - |C|) \\ =& (1-c)(1-\sigma_v). \end{aligned} \quad (6)$$

Applying Equation (6) to Equation (5), we have: $\mathcal{P}_q(v) \leq \frac{d_u}{d_v}\mathcal{P}_q(u) + c(1_{q\in L(v)} - \frac{d_u}{d_v}1_{q\in L(u)}) + (1-c)(1-\sigma_v)$. ■

If we choose u as the pivot and v as one of its clients, we can use u's q-score to bound v's. If a pivot has a low q-score, likely some of its clients can be quickly pruned. Theorem 4 shows that larger σ_v and σ_u lead to better bounds. Pivot-client relations are established as follows: for vertices u and v, if the common 1-hop neighbors take at least σ fraction of each vertex's neighborhood, i.e., $\sigma_u \geq \sigma$ and $\sigma_v \geq \sigma$, we designate either of them as the pivot and the other as the client. Clearly, u and v are within 2-hop of each other. Theorem 4 bounds the q-scores for some of a pivot's 2-hop neighbors. Figure 5 shows that vertices u and v share four 1-hop neighbors in common: $\{a, b, c, d\}$. If $\sigma = 0.5$, either of them can be the pivot of the other. Algorithm 2 shows how to find pivots and their clients.

3) Approximate q-Score Bounding and Pruning: The proposed q-score bounds express the relation between the real q-scores of vertices. However, computing real q-scores is costly for large graphs. Since random walks are used in glceberg to approximate PPVs and approximate q-scores are subsequently computed, will those bounds still be effective for pruning? In this section, we analyze the effectiveness of using approximate q-scores to derive approximate q-score bounds. Our findings are: given a q-score cut-off threshold θ, if approximate q-scores are used to derive approximate q-score bounds as in Corollary 2 and Theorem 4, with some adjustment to θ, the bounds can still be leveraged to prune vertices with a certain accuracy. Specifically, those vertices pruned by those bounds are very likely to be real non-iceberg vertices. The details are in Theorems 5 and 6 and the proofs are in the appendix.

Algorithm 2: Pivot Vertex Selection

Input: G, neighborhood similarity threshold σ
Output: Pivot vertices V_P and their clients
for *Each unchecked v in V* **do**
  Grow v's 2-hop neighborhood $N_2(v)$ using BFS;
  For each unchecked u in $N_2(v)$, check if $N_1(u)$ and $N_1(v)$ satisfy similarity threshold σ; if so, insert u into v's client set, insert v to V_P and mark both v and u as checked;
Return all pivot vertices V_P and their clients;

Theorem 5 (Approximate Neighbor Bound): Let x be an adjacent vertex of vertex v. For a given pruning cut-off threshold θ, let $\theta_1 = \theta - d_v\epsilon/(1-c) + \epsilon$. If $\frac{d_v}{1-c}(\tilde{\mathcal{P}}_q(v) - c1_{q\in L(v)}) < \theta_1 - \epsilon$, where $\tilde{\mathcal{P}}_q(v)$ is the approximate q-score of v using FA approximation with R random walks, then x can be pruned and we have $\Pr[\mathcal{P}_q(x) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$.

Theorem 6 (Approximate PC Bound): Suppose we have vertices u and v and $N_1(u) \cap N_1(v)$ is not empty. Let $\sigma_u = |N_1(u) \cap N_1(v)|/|N_1(u)|$ and $\sigma_v = |N_1(u) \cap N_1(v)|/|N_1(v)|$. For a given pruning cut-off threshold θ, let $\theta_2 = \theta - d_u\epsilon/d_v + \epsilon$. If $\tilde{\mathcal{P}}_q(u)d_u/d_v + c(1_{q\in L(v)} - 1_{q\in L(u)}d_u/d_v) + (1-c)(1-\sigma_v) < \theta_2 - \epsilon$, where $\tilde{\mathcal{P}}_q(u)$ is the approximate q-score of u using FA approximation with R random walks, then v can be pruned and we have $\Pr[\mathcal{P}_q(v) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$.

To summarize, this section shows: (1) Two types of q-score bounds can be used to prune non-iceberg verticess. (2) When random walks are used to approximate q-scores, the bounds become approximate too. However, with certain adjustment to the thresholds and pruning rules, the likelihood for a pruned vertex to be a real non-iceberg vertex can be bounded. We will show later in our experiments that PFA yields good recall in practice. Algorithm 3 shows the workflow of PFA.

Algorithm 3: Pivot Vertex-Based Forward Aggregation

Input: G, query q, threshold θ, neighborhood similarity threshold σ, approximation error ϵ
Output: Graph iceberg vertices
Index all the pivot vertices V_P using σ as in Alg. 2;
for *Each v in V_P* **do**
  Use random walks to get v's approximate PPV, $\tilde{\boldsymbol{p}}_v$;
  Get v's approximate q-score using $\tilde{\boldsymbol{p}}_v$;
  Use approximate q-score bounds to prune vertices using adjusted thresholds based on θ;
for *Each v that is not pruned* **do**
  Use random walks to get v's approximate PPV, $\tilde{\boldsymbol{p}}_v$;
  Get v's approximate q-score using $\tilde{\boldsymbol{p}}_v$;
Return vertices with approximate q-score above $\theta - \epsilon$;

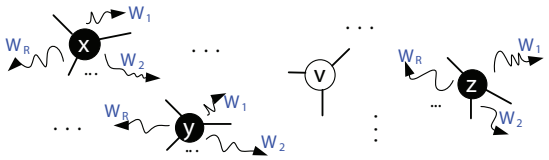


Fig. 6. Backward Aggregation Approximation

V. Backward Aggregation

In this section, we introduce a different aggregation scheme called *backward aggregation* (BA). Instead of aggregating PageRank in a forward manner (adding up the entries of black vertices in a PPV), BA starts from black vertices, and propagates values in their PPVs to other vertices in a backward manner. Specifically, based on the reversibility of random walks, the symmetric property of degree-normalized PPV [19] states that: in an undirected graph G, for any two vertices u and v, the PPVs of u and v satisfy:

$$\boldsymbol{p}_u(v) = \frac{d_v}{d_u}\boldsymbol{p}_v(u), \tag{7}$$

where d_u and d_v are the degrees of u and v, respectively. If we know v's PageRank with respect to u, $\boldsymbol{p}_u(v)$, we can quickly compute the value for its *reverse*, $\boldsymbol{p}_v(u)$, without actually computing v's PPV. For a given query attribute q, the PageRank values of black vertices in any vertex v's PPV are the key to computing v's q-score. In Figure 3(b), BA starts from black vertices, computes their PPVs, and propagates their contributions to the other vertices' q-scores backward (black arrow) according to Equation (7). BA provides a possibility to quickly compute q-scores for the entire vertex set, by starting from only those black vertices. Given that black vertices usually occupy a small portion of V, BA reduces the aggregation time significantly.

A. Backward Aggregation Approximation

Applying BA on approximate PPVs generated by random walks is called *BA approximation*. In Figure 6, for each black vertex x we perform R random walks, $\{W_1, \ldots, W_R\}$, from x to approximate x's PPV. Each walk continues until its first restart. Once such process is done on all the black vertices, for any vertex v in G, v's approximate q-score is the sum of the reverse PageRank scores of the vth entries in the approximate PPVs of the black vertices, computed according to Equation (7). We now analyze the accuracy of such approximate aggregation.

Theorem 7 (BA Approximation): Let $V_q \subseteq V$ be the set of black vertices. Suppose we perform R random walks from each black vertex, x, to approximate its PPV, $\tilde{\boldsymbol{p}}_x$. For any vertex $v \in V$, let $\tilde{\mathcal{P}}_q(v) = \Sigma_{x \in V_q} \frac{d_v}{d_v}\tilde{\boldsymbol{p}}_x(v)$ be the approximate q-score of v using BA. We have $\Pr[\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v) \geq \epsilon] \leq \exp\{-2Rd_v^2\epsilon^2/\Sigma_{x\in V_q} d_x^2\}$ and $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)|] \geq \epsilon] \leq 2\exp\{-2Rd_v^2\epsilon^2/\Sigma_{x\in V_q} d_x^2\}$, where $\epsilon > 0$.

The proof is in the appendix. Now we analyze how well BA retrieves iceberg vertices. As in Algorithm 4, BA retrieves all the vertices whose approximate q-score is above $\theta - \epsilon$ as iceberg vertices. Again we use recall as the measure, which evaluates the percentage of real iceberg vertices that are captured by the BA approximation.

Algorithm 4: Backward Aggregation

Input: G, query q, threshold θ, approximation error ϵ
Output: Graph iceberg vertices
1 **for** *Each black vertex* x **do**
2 Use random walks to get x's approximate PPV, $\tilde{\boldsymbol{p}}_x$;
3 **for** *Each entry* $\tilde{\boldsymbol{p}}_x(v)$ **do**
4 Compute the reverse entry $\tilde{\boldsymbol{p}}_v(x) = \frac{d_x}{d_v}\tilde{\boldsymbol{p}}_x(v)$;
5 Add $\tilde{\boldsymbol{p}}_v(x)$ to v's q-score: $\tilde{\mathcal{P}}_q(v)$;
6 Return vertices with approximate q-score above $\theta - \epsilon$;

Corollary 3 (BA Recall): Given a q-score cut-off threshold θ, for vertex v such that $\mathcal{P}_q(v) \geq \theta$, we have $\Pr[\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon] \geq 1 - 2\exp\{-2Rd_v^2\epsilon^2/\Sigma_{x\in V_q} d_x^2\}$, where $\epsilon > 0$ and $\tilde{\mathcal{P}}_q(v)$ is v's q-score using BA approximation.

Proof: The proof follows from Theorem 7. ■

Therefore the likelihood for a real iceberg vertex v to be retrieved by BA can be bounded. This bound is not as tight as the one for FA. We later show in our experiments that BA achieves good recall in practice, given a reasonable number of random walks. Algorithm 4 describes the BA workflow.

VI. Clustering Property of Iceberg Vertices

Graph iceberg vertices can further be used to discover graph iceberg regions. We achieve this by methods ranging from graph clustering to simple connected component finding. In this section, we describe some interesting properties of how iceberg vertices are distributed in the graph. We discovered that iceberg vertices naturally form connected components surrounding the black vertices in the graph.

A. Active Boundary

Define a *region* $\mathcal{R} = \{V_R, E_R\}$ to be a connected subgraph of G, and the *boundary* of $\mathcal{R}$, $N(\mathcal{R})$, to be the set of vertices such that $N(\mathcal{R}) \cap V_R = \emptyset$ and each vertex in $N(\mathcal{R})$ is directly connected to at least one vertex in V_R. In Figure 7(a), the dark area surrounding region $\mathcal{R}$ forms $\mathcal{R}$'s boundary. Theorem 3 shows that the q-score of a non-black vertex is exactly $(1-c)$ times the average q-score of all its neighbors.

Theorem 8 (Boundary): Given a region $\mathcal{R}$ in G which does not contain any black vertex, if the q-scores of all vertices in $N(\mathcal{R})$ are below the q-score threshold θ, then no vertex in V_R has q-score above θ.

Proof: Equation (4) shows that the q-score of a non-black vertex is lower than the maximum q-score of its neighbors. Suppose there is a vertex $v_0 \in V_R$ such that $\mathcal{P}_q(v_0) > \theta$. Since $\mathcal{R}$ does not contain black vertices, v_0 is non-black, thus at least one of v_0's neighbors has q-score higher than $\mathcal{P}_q(v_0)$. Let it be v_1. The same argument holds for v_1. A path is therefore formed with a strictly increasing sequence of q-scores, and all the q-scores in this sequence are $> \theta$. Since $|V_R|$ is finite, eventually the path goes through $\mathcal{R}$'s boundary, $N(\mathcal{R})$. Since

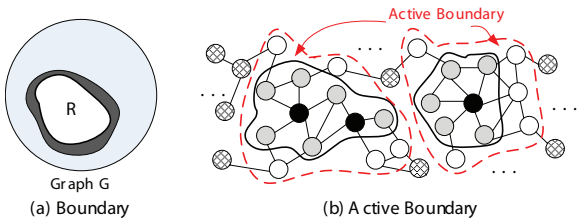


Fig. 7. Boundary and Active Boundary

all vertices in $N(\mathcal{R})$ have a q-score below θ, a contradiction is reached. Therefore no such vertex v_0 exists and all vertices in V_R have q-scores below θ. ■

Corollary 4 (Active Path): If vertex v is an iceberg vertex, there exists a path, called an "active path", from v to a black vertex, that all vertices on that path are iceberg vertices.

Proof: Assume there is an iceberg vertex v_0 (non-black) that can not be linked to a black vertex via such a path. Again we can follow a path starting from v_0 to one of its neighbors, v_1, an iceberg vertex, then to another such neighbor of v_1, and so on. Such a path contains only iceberg vertices. A set of such paths form a region surrounding v_0, containing only iceberg vertices. The boundary of this region only contains vertices with q-scores below θ. The assumption dictates there is no black vertex in this region. This contradicts Theorem 8. So no such vertex v_0 exists. ■

Corollary 5 (Active Region & Active Boundary): Given a query attribute q and q-score threshold θ, all iceberg vertices in G concentrate surrounding black vertices. Each black vertex is surrounded by a region containing only iceberg vertices, which is called "active region". The boundary of such a region is called "active boundary". The q-scores of the vertices in the active boundary are all below θ.

Proof: The proof follows from Corollary 4. ■

Corollary 5 suggests that in order to retrieve all iceberg vertices, we only need to start from black vertices and grow their active regions. The key to grow an active region of a black vertex is to find the active boundary of this region. It is possible that several active regions merge into one if the black vertices are close to each other. Figure 7(b) shows examples of active regions and boundaries. Black vertices contain the query attribute and gray vertices are iceberg vertices. Each black vertex is embedded in an active region containing only iceberg vertices, which is encompassed by a solid black line. All the white vertices between the solid black line and the red dashed line form the active boundaries of those regions. All vertices with grid pattern which are not in any active region have a q-score below the threshold.

Therefore, we have discovered this interesting "clustering" property of iceberg vertices. All iceberg vertices tend to cluster around the black vertices in the graph, which automatically form several interesting iceberg regions in the graph. The size of an iceberg region can be controlled by varying the q-score threshold.

VII. Experiments

gIceberg is evaluated using both real-world and synthetic data. We first conduct motivational case studies on the DBLP network to show that gIceberg is able to find interesting author groups. The remaining experiments focus on: (i) aggregation accuracy; (ii) forward aggregation (FA) and backward aggregation (BA) comparison; (iii) impact of attribute distributions; and (iv) scalability. We observe that: (1) Random walk approximation achieves good accuracy. (2) FA is slightly better than BA in recall and precision, while BA is generally much faster. (3) Pivot vertex selection and q-score bounding effectively reduce runtime. (4) gIceberg is robust to various attribute distributions, and BA is efficient even for dense attribute distribution; (5) BA scales well on large graphs. All the experiments are conducted on a machine that has a 2.5GHz Intel Xeon processor, 32G RAM, and runs 64-bit Fedora 8 with LEDA 6.0 [20]. Figures are best viewed in color.

A. Data Sets

Customer Network (*Customer*). This is a proprietary data set provided by an e-commerce corporation offering online auction and shopping services. The vertices are customers and the edges are their social interactions. The attributes of a vertex are the products that the customer has purchased. This graph has 794,001 vertices, 1,370,284 edges, and 85 product names.

DBLP Network (*DBLP*). This is built from the DBLP[1] repository. Each vertex is an author and each edge represents a co-authorship. The keywords in paper titles are used as vertex attributes. We use a subset of DBLP containing 130 important keywords extracted by Khan et al. [21]. This graph contains 387,547 vertices and 1,443,873 edges.

R-MAT Synthetic Graphs (*R-MAT*). A set of synthetic graphs with power-law degree distributions and small-world characteristics are generated by the GTgraph toolkit[2] using the *Recursive Matrix* (R-MAT) graph model [22]. The vertex number spans across $500K$, $2M$, $4M$, $6M$, $8M$, $10M$. The edge number spans across $3M$, $8M$, $16M$, $24M$, $32M$, $40M$.

TABLE I
Query Attribute Examples

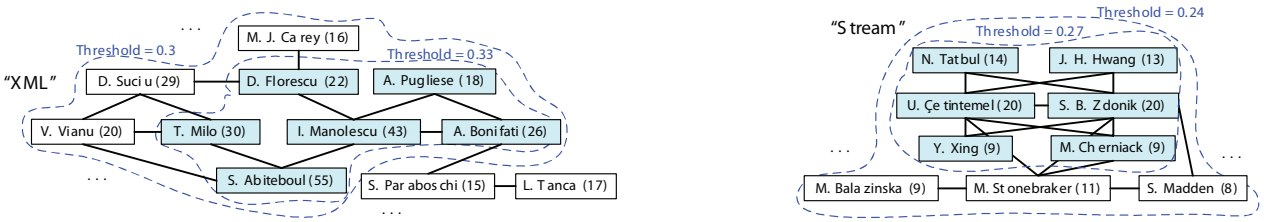
Data Sets	Query Attribute Examples
Customer	"Estée Lauder Eye Cream", "Ray-Ban Sunglasses" "Gucci Glasses", "A&F Women Sweatshirts"
DBLP	"Database", "Mining", "Computation" "Graph", "Classification", "Geometry"

50 queries are used for each graph. Table I shows some query examples for *Customer* and *DBLP*. The attribute generator for *R-MAT* will be introduced in Section VII-E. The q-score threshold is $\theta = 0.5$, if not otherwise specified.

B. Case Study

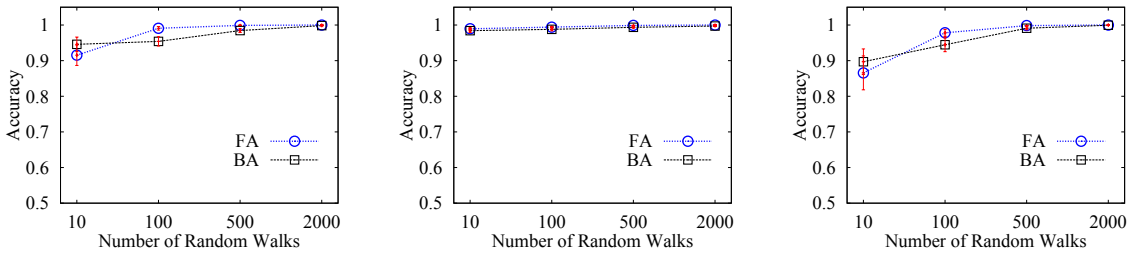
To show that gIceberg finds interesting vertices in a real graph, we conduct a case study on *DBLP*: (1) given a user

[1] www.informatik.uni-trier.de/~ley/db/
[2] http://www.cse.psu.edu/~madduri/software/GTgraph/



(a) Keyword: "XML" (b) Keyword: "Stream"

Fig. 8. Case Studies on *DBLP*



(a) *Customer Subgraph* (b) *DBLP 2010* (c) *R-MAT 3K*

Fig. 9. Random Walk-Based Aggregation Accuracy

specified research topic and an iceberg threshold, we find the iceberg vertices and remove the rest; (2) these vertices form several connected components in the remaining graph. The iceberg vertices imply that many of their neighbors have published in the user specified topic. Therefore, the components shall represent collaboration groups in that area. We will demonstrate that glceberg can indeed discover interesting and important author groups.

Figure 8 shows the top author groups found by glceberg for two query keywords: "XML" and "Stream". The number next to each author's name is the number of his/her publications in that keyword field. All the vertices who have 7+ papers containing the query keyword are retained. There is an edge between two authors if they have collaborated at least 7 times. In glceberg, we set the threshold at a small value and increase it until the author groups become small enough. Take "Stream" for example: the author group size decreases from 9 to 6, as threshold increases from 0.24 to 0.27. For 0.27, the current author group contains 6 authors (in blue). It seems the author groups that glceberg discovers are of high-quality. They are specialized and well-known in the field of "XML" and "Stream". In addition, by varying the q-score threshold, users can easily zoom in and zoom out the author groups and exploit the hierarchical structure with views of multiple granularities. Such zoom in/out effect is not available if we simply use the number of papers as a filter, which will generate many small disconnected components. For example, in Figure 8(b), likely only U. Çetintemel and S. Zdonik will be ranked high, and the others will not show up at all.

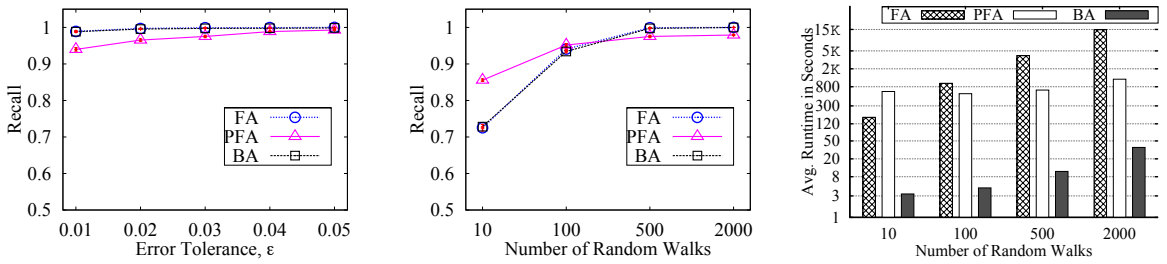
C. Aggregation Accuracy

We now evaluate the accuracy of random walk approximation. We compare random walk-based FA and BA, with the power method-based aggregation, which aggregates over PPVs generated by the power method. We conduct the power iteration until the maximum difference between any entries of two successive vectors is within 10^{-6}. Since the power method is time consuming, this test is done on three small graphs: *Customer Subgraph* is a subgraph of *Customer* with 5,000 vertices and 14,735 edges; *DBLP 2010* is the DBLP network that spans from January 2010 to March 2010, with 12,506 vertices and 19,935 edges; *RMAT 3K* is a synthetic graph generated by the GTgraph toolkit, with 3,697 vertices and 7,965 edges. All three small graphs are treated as independent graphs. *Accuracy* is defined as the number of vertices, whose FA (or BA) approximate q-score falls in between $[-\epsilon, +\epsilon]$ of its q-score computed by the power method, divided by the total number of vertices. We then compute the average accuracy over all the queries. The mean accuracy with standard error bars is shown in Figure 9 ($\epsilon = 0.03$). We vary the number of random walks performed on each vertex. Both FA and BA are shown to empirically produce good accuracy with high mean and low standard error. Since the power method is very slow, hereinafter we apply FA with $2K$ random walks per vertex on larger graphs to provide "ground truth" q-scores.

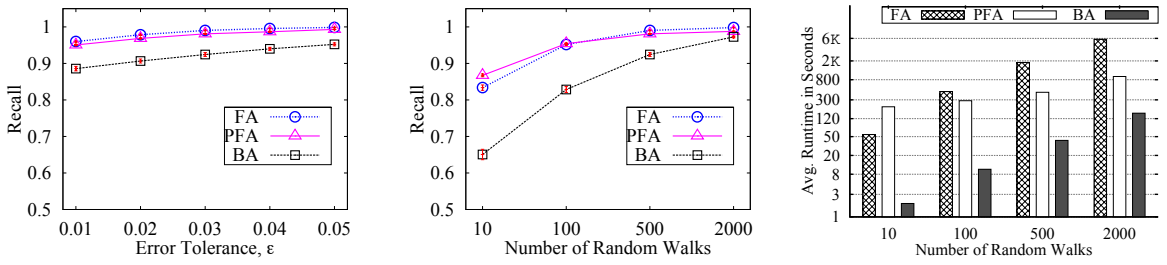
D. Forward vs. Backward Aggregation

We now compare forward and backward aggregation using recall, precision and runtime.

1) Recall and Runtime: Recall is defined as the number of iceberg vertices retrieved by glceberg, divided by the total number of iceberg vertices, i.e., $\big|\{v|\mathcal{P}_q(v) \geq \theta, \tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\big| / \big|\{v|\mathcal{P}_q(v) \geq \theta\}\big|$, where $\mathcal{P}_q(v)$ and $\tilde{\mathcal{P}}_q(v)$ are true and approximate q-scores, respectively. The effectiveness of pivot vertex, approximate q-score bounding and pruning in pivot vertex-based FA (PFA) is also evaluated. In PFA, 150 random

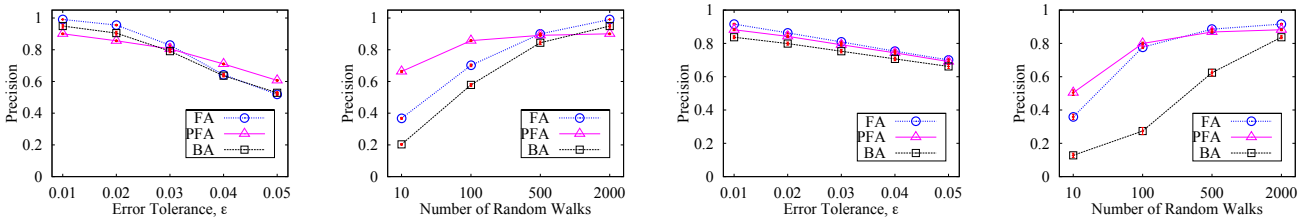


(a) *Customer*, $R = 500$ (b) *Customer*, $\epsilon = 0.03$ (c) *Customer*, Runtime



(d) *DBLP*, $R = 500$ (e) *DBLP*, $\epsilon = 0.03$ (f) *DBLP*, Runtime

Fig. 10. Forward Aggregation vs. Backward Aggregation: Recall and Runtime Comparison



(a) *Customer*, $R = 2000$ (b) *Customer*, $\epsilon = 0.01$ (c) *DBLP*, $R = 2000$ (d) *DBLP*, $\epsilon = 0.01$

Fig. 11. Forward Aggregation vs. Backward Aggregation: Precision Comparison

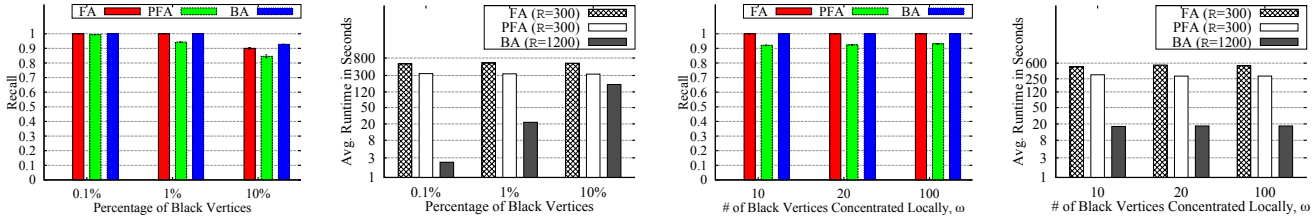
walks are applied on each pivot vertex, while R random walks are applied on the rest. Figure 10 shows the recall and runtime for *Customer* and *DBLP*. We plot the average recall over all queries with error bars on each point. The first column shows how recall changes with ϵ, for $R = 500$; the second column shows how recall changes with R, for $\epsilon = 0.03$. We can observe that: (1) Both FA and PFA yield high recall and BA yields satisfying recall when R is ≥ 500. The approximation captures most of the real iceberg vertices. (2) The standard error across various queries is small for all the methods, showing the performance is consistent and robust to various queries. (3) Recall increases with ϵ and R, which is as expected. (4) It is reasonable that PFA produces better recall than FA when $R \leq 150$, even though PFA uses approximate q-score bounding. This is because for all R values, 150 random walks are always applied on pivot vertices in PFA. Thus when $R \leq 150$, more random walks are used in PFA than in FA.

Runtime comparison in Figure 10 shows that BA significantly reduces the runtime. When R is large, PFA reduces the runtime of FA via pivot vertex and q-score bounding. Since the pivot vertices use 150 random walks, it is expected for PFA to cost more time than FA when R is small. When $R = 100$, PFA is still faster than FA, due to effective pruning. Table II shows that it takes a reasonable amount of offline computation to select pivot vertices. To sum up, BA still yields good recall while reducing the runtime. FA is preferred over BA when higher recall is desired.

TABLE II
PIVOT VERTEX INDEXING COST

Data Sets	*Customer*	*DBLP*	*RMAT 500K*
Time (Hours)	0.176	0.162	1.230
Index/Graph Size (*MB*)	8.48/46.53	4.75/91.68	9.24/53.99

2) Precision: Precision is defined as the number of iceberg vertices retrieved by glceberg, divided by the total number of retrieved vertices, i.e., $\left|\{v|\mathcal{P}_q(v) \geq \theta, \tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\right| / \left|\{v|\tilde{\mathcal{P}}_q(v) \geq \theta - \epsilon\}\right|$. Figure 11 shows the curves of the average precision over all queries with error bars for *Customer* and *DBLP*. We can see that: (1) FA and PFA yield better precision than BA. When $R = 2000$, all the methods yield decent precision. (2) The standard error is small for all the methods, showing the performance is consistent across queries. (3) Precision decreases with ϵ and increases with R



(a) Recall vs. Percentage (b) Runtime vs. Percentage (c) Recall vs. Skewness (d) Runtime vs. Skewness

Fig. 12. Attribute Distribution Test

as expected. (4) As previously analyzed, it is reasonable for PFA to produce better precision than FA when $R \leq 150$.

We can see that BA is much faster than FA and BA yields good recall when $R \geq 500$. However, the precision of BA is not satisfactory unless $R \geq 2000$. FA overall yields better precision than BA. Therefore a fast alternative to achieve both good recall and precision would be: (1) apply BA to retrieve most real iceberg vertices with good recall; and then (2) apply FA on those retrieved vertices to prune out the "false positives" to further achieve good precision.

E. Attribute Distribution

We now test the impact of attribute distribution on the aggregation performance. To customize the attribute distribution, a synthetic *R-MAT* graph with 514,632 vertices and 2,998,960 edges is used (*R-MAT 500K*). Two tests are done: (1) We customize the *percentage* of the black vertices, which is computed as $|V_q|/|V|$. Given a query attribute q, we randomly distribute it into the graph. The set of black vertices is V_q. (2) We customize the *skewness* of the black vertex distribution. The attribute q can be randomly dispersed without any specific patterns, or concentrated in certain regions. To instantiate this idea, we randomly select a set of root vertices, and randomly assign q to a certain number, ω, of vertices within each root's close neighborhood. Let $|V_r|$ be the total number of roots. We have $\omega * |V_r| = |V_q|$. If $|V_q|$ is fixed, by tuning ω and $|V_r|$, we can control the skewness of the attribute distribution. A higher ω indicates a higher skewness. We set $\epsilon = 0.05$, $R = 300$ for FA and PFA, and $R = 1200$ for BA.

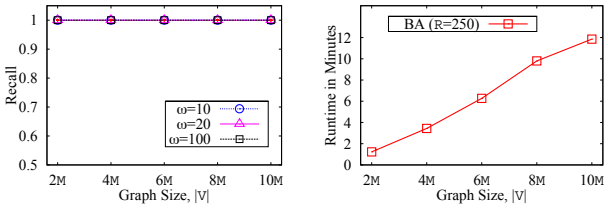
For percentage test, 50 queries are randomly generated for each percentage. Figure 12(a) plots the mean recall with standard error. The percentage varies from 0.1% to 10%. All three methods yield good recall with small standard errors. The recall slightly decreases when the percentage increases. Figure 12(b) shows that the runtime of BA increases with $|V_q|/|V|$, which is as expected. BA is much faster, even when its random walk number is four times that of FA and PFA.

For skewness test, 50 queries are randomly generated for each ω. Figure 12(c) plots the mean recall with standard error. The number of black vertices concentrated locally surrounding each root vertex, ω, changes from 10 to 100. The black vertex number is $|V_q| = 50K$. All the methods yield good recall with small standard errors. PFA yields slightly worse recall, due to approximate q-score bounding and pruning. Figure 12(d) shows the BA runtime is almost constant because $|V_q|$ is constant and BA is significantly faster.

These figures show that FA/BA are not sensitive to the skewness in terms of recall and runtime. BA is sensitive to the percentage of black vertices in terms of runtime, but not sensitive in terms of recall.

F. Scalability Test

As shown in previous experiments, BA is much more efficient than FA and PFA. We further demonstrate how scalable BA is on large graphs. A set of *R-MAT* synthetic graphs with $|V| = \{2M, 4M, 6M, 8M, 10M\}$ and $|E| = \{8M, 16M, 24M, 32M, 40M\}$ are generated. The percentage of black vertices is 0.5% for all. The skewness of the attribute distribution, ω, changes from 10 to 100. We set $\epsilon = 0.05$ and $R = 250$ for BA. Figure 13(a) plots the mean recall of BA with standard errors across all the queries. BA yields good recall on all the graphs and the recall is not sensitive to attribute distribution skewness. Figure 13(b) shows that the runtime of BA is approximately linear to the graph size. In conclusion, BA exhibits good scalability over large graphs. FA and PFA do not scale as well as BA. It takes them a few hours to return on large graphs. Therefore, we consider BA as a scalable solution for large graphs with decent recall. If higher recall is desired, users can choose FA instead of BA.



(a) Recall (b) Runtime

Fig. 13. BA Scalability Test

VIII. Related Work

Graph iceberg analysis is related to the following areas.

Iceberg cube and graph OLAP. In multidimensional OLAP [23], an iceberg cube contains cells whose measure satisfies a threshold [4], [23]. Existing iceberg cubing methods include top-down methods, bottom-up methods, integration methods [23], [24], etc. Iceberg analysis on graphs has

been under-explored due to the absence of dimensionality in graph data. The first work to place graphs in a rigid multi-dimensional and multi-level framework is [4]. The objective and results in [4] are substantially different from those in this paper. With a distinct focus on iceberg analysis, we define the concept of graph icebergs and propose scalable solutions.

Graph aggregation. Graph aggregation is to summarize or aggregate networks to form a hierarchy. `SNAP` operations were introduced in [2] to consistently merge nodes and edges with respect to a predefined hierarchy. When such hierarchy is unknown, graph summarization can be achieved via graph clustering [25]. A local neighborhood aggregation framework was proposed in [5], which finds the top-k vertices with the highest aggregation values over their neighbors. Our work extends [5] by incorporating local proximities to the target attribute, into the aggregation.

Graph anomaly detection. Anomaly detection has been studied in graph-based data [3], [26], [27]. [26] proposed both anomalous substructure detection and anomalous subgraph detection. [3] discovered several new rules in density, weights, ranks and eigenvalues that govern local neighborhoods and used these rules for anomaly detection. glceberg instead focuses on retrieving graph vertices interesting to a certain query via aggregation. There is a distinct difference between graph anomaly and graph iceberg. However, the anomaly score of a vertex can be treated as its attribute value. Thus, graph anomaly detection can be used in glceberg to find icebergs with many abnormal close neighbors.

Densest subgraph finding. In traditional densest subgraph studies, the subgraph density is defined as the average vertex degree of the subgraph [28], [29]. The densest k-subgraph (DkS) problem finds the densest subgraph of k vertices, which is NP-hard [30]. Most of these studies only consider graph connectivity. [6] introduces the novel problem of finding cohesive patterns. A cohesive pattern is a connected subgraph whose density exceeds a given threshold and has homogeneous feature values. What makes glceberg different from [6] is that the iceberg regions discovered via clustering iceberg vertices include, but not limited to, dense regions. If an edge-sparse region contains a high percentage of black vertices, it will likely form an iceberg region as well, since the vertices within will have high q-scores.

Local clustering. Local clustering finds a cluster containing a given vertex without looking at the whole graph [12], [31]. A core method called `Nibble` was proposed in [31]. By using the personalized PageRank [11] to define the nearness, [12] introduced an improved version, `PageRank-Nibble`. glceberg can be combined with standard graph clustering techniques to find iceberg regions. By aggregating over PPVs to define the nearness between a vertex and an attribute, glceberg considers both vertex attributes and edge connectivities.

IX. Conclusions

This paper introduces a novel concept, graph iceberg, which extends iceberg queries to vertex-attributed graphs and identifies interesting vertices that are close to an attribute using an aggregate score. A scalable framework, glceberg, is proposed with two aggregation schemes: forward and backward aggregation. Optimization techniques are designed to prune unpromising vertices and reduce aggregation cost. Experiments on real-world and synthetic large graphs show the effectiveness and scalability of glceberg. Future work includes finding icebergs incrementally on time-evolving graphs, and discovering other types of icebergs such as paths, trees, and subgraph patterns.

Acknowledgment

This research was sponsored in part by NSF IIS-0905084 and by the Army Research Laboratory under cooperative agreements W911NF-09-2-0053 and W911NF-11-2-0086. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

[1] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *ICDM*, 2002, pp. 721–724.
[2] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *SIGMOD*, 2008, pp. 567–580.
[3] L. Akoglu, M. McGlohon, and C. Faloutsos, "oddball: Spotting anomalies in weighted graphs," in *PAKDD*, 2010, pp. 410–421.
[4] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, "Graph OLAP: a multi-dimensional framework for graph data analysis," *Knowl. Inf. Syst.*, vol. 21, no. 1, pp. 41–63, 2009.
[5] X. Yan, B. He, F. Zhu, and J. Han, "Top-k aggregation queries over large networks," in *ICDE*, 2010, pp. 377–380.
[6] F. Moser, R. Colak, A. Rafiey, and M. Ester, "Mining cohesive patterns from graphs with feature vectors," in *SDM*, 2009, pp. 593–604.
[7] A. Prakash, N. Valler, D. Andersen, M. Faloutsos, and C. Faloutsos, "Bgp-lens: patterns and anomalies in internet routing updates," in *KDD*, 2009, pp. 1315–1324.
[8] P. Bogdanov, M. Mongiov, and A. Singh, "Mining heavy subgraphs in time-evolving networks," in *ICDM*, 2011, pp. 81–90.
[9] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman, "Computing iceberg queries efficiently," in *VLDB*, 1998, pp. 299–310.
[10] T. L. Fond and J. Neville, "Randomization tests for distinguishing social influence and homophily effects," in *WWW*, 2010, pp. 601–610.
[11] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Stanford InfoLab, Technical Report, 1999.
[12] R. Andersen, F. R. K. Chung, and K. J. Lang, "Local partitioning for directed graphs using PageRank," *Internet Mathematics*, vol. 5, no. 1, pp. 3–22, 2008.
[13] M. Franceschet, "PageRank: Standing on the shoulders of giants," *Commun. ACM*, vol. 54, no. 6, pp. 92–101, 2011.
[14] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized PageRank," *PVLDB*, vol. 4, no. 3, pp. 173–184, 2010.
[15] P. Berkhin, "Bookmark-coloring approach to personalized PageRank computing," *Internet Mathematics*, vol. 3, no. 1, 2006.
[16] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized PageRank: Algorithms, lower bounds, and experiments," *Internet Mathematics*, vol. 2, no. 3, 2005.
[17] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statistical Assoc.*, vol. 58, no. 301, pp. 13–30, 1963.
[18] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003, pp. 271–279.
[19] P. Sarkar and A. W. Moore, "Fast nearest-neighbor search in disk-resident graphs," in *KDD*, 2010, pp. 513–522.
[20] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[21] A. Khan, X. Yan, and K.-L. Wu, "Towards proximity pattern mining in large graphs," in *SIGMOD*, 2010, pp. 867–878.
[22] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *SDM*, 2004.
[23] K. S. Beyer and R. Ramakrishnan, "Bottom-up computation of sparse and iceberg cubes," in *SIGMOD*, 1999, pp. 359–370.
[24] D. Xin, J. Han, X. Li, and B. W. Wah, "Star-cubing: Computing iceberg cubes by top-down and bottom-up integration," in *VLDB*, 2003, pp. 476–487.
[25] A. Y. Wu, M. Garland, and J. Han, "Mining scale-free networks using geodesic clustering," in *KDD*, 2004, pp. 719–724.
[26] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *KDD*, 2003, pp. 631–636.
[27] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *ICDM*, 2005, pp. 418–425.
[28] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," in *APPROX*, 2000, pp. 84–95.
[29] S. Khuller and B. Saha, "On finding dense subgraphs," in *ICALP (1)*, 2009, pp. 597–608.
[30] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, "Detecting high log-densities: An $O(n^{1/4})$ approximation for densest k-subgraph," in *STOC*, 2010, pp. 201–210.
[31] D. A. Spielman and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning," *CoRR*, vol. abs/0809.3232, 2008.

APPENDIX

Proof of Theorem 1:

Proof: For each random walk $W_i(i = \{1, \ldots, R\})$, let X_i be a random variable such that

$$X_i = \begin{cases} 1, & \text{if random walk } W_i \text{ ends at vertex } x, \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\tilde{\boldsymbol{p}}_v(x) = (\Sigma_i X_i)/R$. According to the findings in [16], $\boldsymbol{p}_v(x) = E[\tilde{\boldsymbol{p}}_v(x)]$. From the definition of X_i, we have $\Pr[X_i \in [0,1]] = 1$, and $X_i(i = \{1, \ldots, R\})$ are independent variables, therefore according to Hoeffding Inequality, we can derive

$$\Pr[\frac{\Sigma_i X_i}{R} - E[\frac{\Sigma_i X_i}{R}] \geq \epsilon] \leq \exp\{-\frac{2(R\epsilon)^2}{\Sigma_{i=1}^{i=R}(1-0)^2}\} \leq \exp\{-2R\epsilon^2\}.$$

Therefore, we have $\Pr[\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x) \geq \epsilon] \leq \exp\{-2R\epsilon^2\}$ and $\Pr[|\tilde{\boldsymbol{p}}_v(x) - \boldsymbol{p}_v(x)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$. ■

Proof of Theorem 2:

Proof: Recall that a vertex which contains attribute q is called a black vertex. For each random walk $W_i(i = \{1, \ldots, R\})$, let Y_i be a random variable such that

$$Y_i = \begin{cases} 1, & \text{if random walk } W_i \text{ ends at a black vertex,} \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\tilde{\mathcal{P}}_q(v) = (\Sigma_i Y_i)/R$ and $\mathcal{P}_q(v) = E[(\Sigma_i Y_i)/R]$. According to the definition of Y_i, $\Pr[Y_i \in [0,1]] = 1$, and $Y_i(i = \{1, \ldots, R\})$ are independent variables. Therefore, according to Hoeffding Inequality, we can derive

$$\Pr[\frac{\Sigma_i Y_i}{R} - E[\frac{\Sigma_i Y_i}{R}] \geq \epsilon] \leq \exp\{-\frac{2(R\epsilon)^2}{\Sigma_{i=1}^{i=R}(1-0)^2}\} \leq \exp\{-2R\epsilon^2\}.$$

Similarly, we have $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)| \geq \epsilon] \leq 2\exp\{-2R\epsilon^2\}$. ■

Proof of Theorem 5:

Proof: Since $\frac{d_v}{1-c}(\tilde{\mathcal{P}}_q(v) - c1_{q\in L(v)}) < \theta_1 - \epsilon$, we have $\tilde{\mathcal{P}}_q(v) < \frac{(1-c)(\theta_1-\epsilon)}{d_v} + c1_{q\in L(v)}$. From Theorem 2, we know $\Pr[\mathcal{P}_q(v) - \epsilon \leq \tilde{\mathcal{P}}_q(v) \leq \mathcal{P}_q(v) + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Thus $\Pr[\mathcal{P}_q(v) \leq \tilde{\mathcal{P}}_q(v) + \epsilon < \frac{(1-c)(\theta_1-\epsilon)}{d_v} + c1_{q\in L(v)} + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Based on $\theta_1 = \theta - d_v\epsilon/(1-c) + \epsilon$, we can derive $\Pr[\frac{d_v}{1-c}(\mathcal{P}_q(v) - c1_{q\in L(v)}) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. So from Corollary 2, $\Pr[\mathcal{P}_q(x) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. ■

Proof of Theorem 6:

Proof: Let $B = c(1_{q\in L(v)} - 1_{q\in L(u)}d_u/d_v) + (1-c)(1-\sigma_v)$. Since $\tilde{\mathcal{P}}_q(u)d_u/d_v + c(1_{q\in L(v)} - 1_{q\in L(u)}d_u/d_v) + (1-c)(1-\sigma_v) < \theta_2 - \epsilon$, we have $\tilde{\mathcal{P}}_q(u) < \frac{d_v(\theta_2-\epsilon-B)}{d_u}$. From Theorem 2, we know $\Pr[\mathcal{P}_q(u) - \epsilon \leq \tilde{\mathcal{P}}_q(u) \leq \mathcal{P}_q(u) + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Thus $\Pr[\mathcal{P}_q(u) \leq \tilde{\mathcal{P}}_q(u) + \epsilon < \frac{d_v(\theta_2-\epsilon-B)}{d_u} + \epsilon] \geq 1 - 2\exp\{-2R\epsilon^2\}$. Based on $\theta_2 = \theta - d_u\epsilon/d_v + \epsilon$, we can derive $\Pr[\mathcal{P}_q(u)d_u/d_v + B < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. So from Theorem 4, $\Pr[\mathcal{P}_q(v) < \theta] \geq 1 - 2\exp\{-2R\epsilon^2\}$. ■

Proof of Theorem 7:

Proof: For any vertex v, for each random walk $W_i^x(i = \{1, \ldots, R\}, x \in V_q)$ starting from black vertex x, let Z_i^x be a random variable such that

$$Z_i^x = \begin{cases} 1, & \text{if random walk } W_i^x \text{ ends at a } v, \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $\tilde{\boldsymbol{p}}_x(v) = (\Sigma_i Z_i^x)/R$. $\boldsymbol{p}_x(v) = E[(\Sigma_i Z_i^x)/R]$. By Equation (7), we have $\tilde{\boldsymbol{p}}_v(x) = \frac{d_x}{d_v}(\Sigma_i Z_i^x)/R$ and $\boldsymbol{p}_v(x) = E[\frac{d_x}{d_v}(\Sigma_i Z_i^x)/R]$. We therefore can further derive

$$\tilde{\mathcal{P}}_q(v) = \Sigma_{x\in V_q}\tilde{\boldsymbol{p}}_v(x) = \Sigma_{x\in V_q}\big(\frac{d_x}{d_v}(\Sigma_i Z_i^x)/R\big) = \Sigma_{x\in V_q}\Sigma_i\frac{d_x Z_i^x}{d_v R},$$

$$\mathcal{P}_q(v) = E[\Sigma_{x\in V_q}\boldsymbol{p}_v(x)] = E[\Sigma_{x\in V_q}\Sigma_i\frac{d_x Z_i^x}{d_v R}].$$

Let $A_i^x = \frac{d_x Z_i^x}{d_v R}$. A_i^x is a random variable such that $\Pr[A_i^x \in [0, \frac{d_x}{d_v R}]] = 1$ and $A_i^x(i = \{1, \ldots, R\}, x \in V_q)$ are independent from each other. We have $\tilde{\mathcal{P}}_q(v) = \Sigma_x\Sigma_i A_i^x$ and $\mathcal{P}_q(v) = E[\Sigma_x\Sigma_i A_i^x]$. According to Hoeffding Inequality,

$$\Pr[\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v) \geq \epsilon] = \Pr[\Sigma_x\Sigma_i A_i^x - E[\Sigma_x\Sigma_i A_i^x] \geq \epsilon] \leq \exp\{-\frac{2\epsilon^2}{\Sigma_{x\in V_q}\Sigma_{i=1}^{i=R}\frac{d_x^2}{d_v^2R^2}}\} \leq \exp\{-\frac{2Rd_v^2\epsilon^2}{\Sigma_{x\in V_q}d_x^2}\}.$$

Similarly $\Pr[|\tilde{\mathcal{P}}_q(v) - \mathcal{P}_q(v)|] \geq \epsilon] \leq 2\exp\{-\frac{2Rd_v^2\epsilon^2}{\Sigma_{x\in V_q}d_x^2}\}$. ■