Pirouz and Zhan *J Big Data* (2016) 3:12
DOI 10.1186/s40537-016-0047-2

Journal of Big Data

RESEARCH Open Access

Optimized relativity search: node reduction in personalized page rank estimation for large graphs

Matin Pirouz* and Justin Zhan

*Correspondence:
matin.pirouznia@unlv.edu
Department of Computer Science, University of Nevada Las Vegas, 4505 Maryland Parkway, Las Vegas, NV 89154-4026, USA

Abstract

This paper proposes an algorithm called optimized relativity search to reduce the number of nodes in a graph when attempting to decrease the running time for personalized page rank (PPR) estimation. Even though similar estimations have been done, this method significantly increases the speed of computation, making it a feasible candidate for large graph solutions, such as search engines and friend recommendation techniques used in social media. In this study, the weighted page rank method was combined with the Monte-Carlo technique and a local update algorithm over a reduced map space; this algorithm was developed to achieve a more accurate and faster search method than FAST PPR. The experimental results showed that for nodes with a high degree of incoming nodes, the speed of estimation was twice as fast compared to FAST PPR, at the expense of a little accuracy.

Keywords: Estimation, Graph reduction, Node reduction, PageRank

Introduction

The page rank (PR) algorithm was first used by Google to rank web pages. Because today's complex social networks need a fast and scalable algorithm for their searching needs, a great many new PR algorithms utilize improved versions of the initial PR mechanism. PR has found usage in a variety of applications, including such social networks as Twitter recommendation systems, and scientific data bases that search for the relative importance of publications, for example [1–4]. This algorithm relies on the findings of the very first repository of web pages, called WebBase, which was an experimental prototype used as a proof of concept for PR [5]. Other studies included research on URL's and web search engines by Chakrabati et al. [6]; as well as work on popularity-oriented PR and the prototype search engine by Bharat and Mihaila [7].

Later, an algorithm called personalized page rank (PPR) was introduced that was different in terms of attributes used to weigh the node [8, 9]. In PR, pages are considered as nodes, and a linear formula calculates the relation between a page and every other related page in the network. However, PPR computes base connection values and the weight of nodes based on such attributes as the inter-node relationship, the related title of search that connects them, and the current connection types between them. This feature makes PPR more accurate than PR, and returns a result according to neighboring

nodes rather than every existing node of the network. Nevertheless, the problem it causes is that it works slower and causes bottlenecks, particularly as the network grows to include bigger nodes. Part of PPR is based on random walks and part is based on a local update in parallel. Random walks have been used before, specifically using a sampling of data to calculate statistics from the Internet [10]. In addition, they have been used to find similarity values [11].

PR and PPR values are very similar but their difference lies in whether or not the proposed probability vector over vertices is unified for all the vertices and/or each node. If there is such a unified number, then PR is used to compute the global probability; however, if they are dependent of the nodes, PPR is used to calculate a node rank from only a single vertex aspect [12, 13].

Importance of a given node is, in fact, the accumulative importance of the incoming node edges to it. For example, the importance of edge '*e*' is dependent on the number on outgoing edges from node '*v*'. The number of nodes plays an important role on a bidirectional algorithm as it is a 'node base', meaning that the bigger the number of nodes get, the more the time complexity increases. One method used in this study was to reduce number of nodes used by applying a similarity algorithm, which finds similar nodes and combines them together as a single node. There is a threshold choosing the nodes to be considered as similar nodes. If the number of nodes in the graph between node *I* and node *E* are reduced, then the number of iterations needed to reach the target *E* also will reduce. On the other hand, the number of random walks needed to reach to the front set will decrease as well.

PPR is a localized method of estimation in a graph, based on the notions defined in PR. However, PPR is used for calculating the distance between two nodes in a given graph. Based on the calculated value, how close two nodes are to one another can be determined. The PR value, which is an accumulated number of all common nodes between a pair of nodes, can be used in such applications as friend recommendations on Facebook or Twitter. TwitterRank optimizes based on a follower-based graph [14]. Although PPR estimation has been performed by using various methods, it still is a challenge because it has an online query-based application and can be very demanding when it comes to resources [15].

The motivation for this study stemmed from determining that it is possible to decrease the estimation time of search in social media using a complexity estimation function by applying node-reduction models. According to Lofgren et al. [16], if a user is looking for some nodes in Twitter, and a node a node of particular interest has many followers, then it will take more time to estimate the chances of getting the right result than if the user is looking for a node having fewer followers. This is because a bidirectional algorithm works using two main methodologies: The first one, the random walk, picks random stats and travels through them to reach the given end node. In this method, the probability to hit the target is $1/n$ as there are n nodes in each step that can be hit. So, reducing nodes to k nodes will give a probability of $1/(k = n - m)$, where m is the number of nodes removed from the graph.

The second part of the program is based on an existing method called the local update algorithm [17]. In this method, traveling between nodes happens with order. From a starting node, the walker goes through every and all the nodes and their children. In each step of changing position from node to the next node, three parameters are calculated: probability, residual error, and moving massage. This iteration will happen until

the walker reaches some certain threshold. At that point, several calculations are made to estimate the PPR value. However, as the number of nodes in the graph gets bigger, the queries will demand more resources, such as the space needed to do the calculation for every node in the graph. An optimized search method saves the space needed for calculating m iterations for every following node of n.

The rest of this paper is organized as follows. "Terms and definitions" section summarizes the preliminaries of the paper. Related works are explained in "Related work" section. The proposed algorithm and its characteristics and descriptions are outlined in "Approach" section. "Experimental results" section discusses the experimental results. Concluding remarks and future research are made in "Conclusion" section.

Terms and definitions

Terminology

In this section, all the basic functions and notations used in this paper are defined. The underlying directed graph G (V; E) is given, and each node and edge are designated by V and E, respectively. Matrix A is the adjacency matrix. The letter 'u' is used to designate the node, or web page, currently positioned upon, and a set of web pages is designated by 'U'. The symbol 'α' denotes the probability of moving from a node to a neighboring node (transition), and $(1-\alpha)$ is denotes the probability of moving to a non-neighboring node (teleportation), as shown in (1):

$$P = (1-\alpha)\cdot \mathrm{PA} + \alpha \mathrm{r} \tag{1}$$

where 'r' is the probability vector distributed over U set of web pages and P is the PR vector solved by (1). The edge between each pair of nodes in a directed graph is shown with 'd_{in}' and 'd_{out}' for incoming and outgoing edges, respectively. The algorithm discussed in this work are estimated over a pair of nodes which are i for initiating node and 'e' for the end node. Here, 'D' is the diagonal matrix, and 'C' is the covariance matrix. The eigenvector is shown by 'I' and eigenvalues are denoted by 's'. The next step is to apply the local update and random walk.

In order to find PR using the iteration method, the eigenvector and eigenvalues for the graph matrix. And in order to find the eigenvector, three conditions must be met:

1. In order to have a stochastic matrix which means it is non-negative and the sum of each row is one, 'q' represents the proposed stochastic matrix.
2. In order to have an irreducible graph, that means it must be graphed to make it possible to get from any node to another one directly or through other nodes.
3. The graph needs to be aperiodic.

The oldest method used to solve the PR problem is called the Power Iteration algorithm. This is a simple method, but it has slow convergence so is not a good solution for big matrices.

Local algorithm

The local update algorithm [17] calculates the estimation value of neighboring nodes to the specified node u. The effect of a neighbor(s) on the end node is computed using

these estimations. The PR is calculated and presented as a vector of values, and specifies how important each neighbor is for the PR estimated for the given node. The main component to be computed in the local update algorithm is an estimation set, which is an accumulative estimation of neighboring nodes from all the local nodes to the node u with respect to a given threshold, which helps to count how far each node is from the target node u. A local update list of estimations is known as the contribution list.

The formula of contribution vector P' is as follows:

$$P' = Ppr(\alpha, v) \quad if\ P \geq 0 \tag{2}$$

The local update algorithm consists of several iterating steps, which are calculated as sequence of calculations. The set of operations taken is called pushback operations [17]. Time complexity of the algorithm depends on incoming nodes to node u rather than the entire iteration time, as the estimation is based on a single initiation node:

$$p \leq \frac{\min(Pr_{\alpha}(v), P_{\max})}{\alpha \in} + 1 \tag{3}$$

Based on work by Anderson et al. [17], the algorithm for the pushback function is:

Algorithm 1 Push Back Calculation [17]

Let $\mathbf{p}' = \mathbf{p}$ and $\mathbf{r}' = \mathbf{r}$, except for these changes:

1. $\mathbf{p}'(u) = \mathbf{p}(u) + \alpha\mathbf{r}(u)$.
2. $\mathbf{r}'(u) = 0$.
3. For each vertex w such that $w \rightarrow u$:

$$\mathbf{r}'(w) = \mathbf{r}(w) + (1 - \alpha)\mathbf{r}(u)/d_{out}(w).$$

Reference [17] defines the PR estimation function as:

Algorithm 2 Estimation Value Calculation [17]

Given v, α, ϵ, P_{max}

Let p = 0, and r = e_v.

2. While r(u) > ϵ for some vertex u:
   (a) Pick any vertex u where r(u) ⩾ϵ.
   (b) Apply pushback (u).
   (c) If ||p||1 ⩾ p_{max}, halt and output = p.
3. **Output** C = p.

The original method defines two considerable time complexity calculations: (a) the first one is the upper bound; and (b) the second one is the lower bound, which happens when the calculation of the target node is not given for the value. Considering the fact that the PR value of a certain node is accumulative PR values from its most significant neighbors, we can calculate the lower bound of node u by adding the estimation of the top neighboring nodes k. So p should be less than $pr_\alpha(u)$ as shown in (4):

$$P_k(1+\delta)^{-2} \leq P \leq pr_\alpha(v) \tag{4}$$

The notations throughout this paper are listed in Table 1.

Related work

PR methods used for page rank calculations come in a different variety of forms. The following are the bases for the proposed algorithm in this paper. The first work ever done on PR was by using the Naive Method, which is also well-known as the PR method [18]. This method is able to personalize any page with the power iteration in query time; however, it comes with the price of being infeasible to serve online personalization.

The Topic-Sensitive PR [19] method is restricted to linear combination of n topics, and is limited in scalability. However, a good point about this method is that it does distributed computing. The Jeh and Widom method [20], also known as Hub Decomposition, was a magnificent change but it is restricted to personalization on the top U-ranked pages to a total number less than a hundred thousand nodes. The problem with this method is the space needed to store processing data. In fact, space needed for the under process data is twice the original data size.

The block rank method [21] does personalization on hosts, and it is limited in scalability as it power-iterates in query time. On the positive side, it reduces the number of iterations and has distributed computing. The fingerprint method [22] personalizes any pages

Table 1 Notations and descriptions

Notation	Description
N	Number of nodes in the graph
K	Number of nodes after reduction
M	Number of iteration for a certain node
V, E	Presentation of nodes and edge
$\{w, u\}, U$	Designated node, Set of u's
$\alpha, (1-\alpha)$	Transition, Teleportation probability
r, P'	Probability, contribution vector
q	Stochastic matrix
d_{in}, d_{out}	Incoming and outgoing edges
A, D, C	Adjacency, diagonal, and covariance matrices, respectively
δ	Given threshold
ϵ	Error parameter
t	Iteration number
x	Estimated number for random walk
R	Reduced space matrix
C	Data matrix

and has no limits on scalability as it works with a linear-size storage and has distributed computation; however, it is not precise enough, and gives only an approximation.

The local update method [17] is proven and exact on approximation, but becomes too slow when it comes to big graphs and nodes with many followers. The condition with a random surfer [23], also known as the Monte Carlo model, has been out there for a long time but it is not sufficient in time when it comes to big graphs. Method of personalized edge weight [24] is fast; however, there is the cost of pre-calculation and it works only on edge-weighted graphs. This method is based on the weighted PR algorithm discussed by Xing and Ghorbani [25]. Several other acceleration methods for PR exist, such as the uniform approach proposed by McShertty [12] which is based on the base computation efficiency used by Haveliwalian [26].

Zhu et al. [27] introduced a more computationally-efficient and scalable hub-based model of PPR based on scheduled approximation. This model achieved near-constant time irrespective of graph size. However, their scheme assumes pre-determined values which can prove challenging for dynamic graphs and re-computation takes much resource. The DRAGON algorithm, introduced by Tong et al. [28] in 2011, uses an optimization concept to find diversified top-k ranking list for large graphs. DRAGON achieves a linear model with respect to graph size. An older moderl, HubRank, is a search system in entity-relation graphs developed by Chakrabarti [29] in 2007. The main difference between this work and the previous works is on the type of data used. This work lacked the theoretical guarantee of the result score. Peter Lofgren et al. [30] introduced a modification of their original algorithm [16] using a bidirectional perspective. In their most recent work, they have used a parallel approach in order to increase the speed of the estimation and social search, with results of up to 8 times over existing estimators. It should be noted that the findings of our propose algorithm can be applied to such novel approaches, achieving even faster results.

Monte Carlo

Monte Carlo is an estimation method used to estimate different properties of sample populations. During PPR calculations, various nodes and values are used as samples. An estimator is calculated for every one of them, and then a range of different calculated estimations for every sample group is used, called the distribution group over the sample set. In every round of experimentation, there are slightly different values from estimating each sample, known as the sampling error. An expectation value usually is used closest to the highest frequency estimator accruing in the estimation. In the Monte Carlo method, a predefined population is used with defined parameters α, β, and ϵ as the error parameters. Estimation results from several experiments help in the process of finding out if the estimation is biased or unbiased [31–33].

In this estimation, Monte Carlo is used with random walk to simulate the act of different user surfing around the web or social media to other pages or other people profile. The way random walks work is like there are always two points in which the user want to exchange place. To calculate the probability that the walker will go from initiating point to the next point considering the factor of α is equal to.

$$x^{(t+1)} = \sum_{t=1}^{t=n} x^t + x \tag{5}$$

FAST-PPR

FAST_PPR analysis is a bidirectional algorithm that estimates the probability that a single walk from an initiation node i can end up on the end node e, with the time complexity of $\frac{1}{\sqrt{\delta}}$ as δ a given threshold.

δ is the threshold for recommendable nodes. As a result, PPR always will be greater than δ. As already stated, the estimated value can be used as a reference for a friend's recommendation in social media or as advertisement in the web. In the real world, this estimation is calculated from a single node to thousands of target nodes. However, in FAST_PPR, the case was investigated only from node i to the end node e in order to simplify the processes. Processing time depends on graph size. Estimation may take time in the range of δ to n, whereas as δ is the threshold and n is the number of nodes in the graph. So, if the estimated value of the probability from node i to node e is less than δ, then it means there is not a single walk from the initiation node, to the end node and the recommendation should not happen.

The second part of this algorithm works based on push, or the local update algorithm. The walker will travel from the end node e to every entering node to e, and calculate an estimation of probability of reaching e from the new node u with a single walk. These processes will take a time complexity of an upper bound equal to $\frac{a}{\delta}$ where a is the average number of edges over the nodes in a given graph.

In the local update algorithm, which is a part of the proposed method, the target set consists of the nodes with a probability higher than δ to hit the end node. The set frontier consists of the nodes that do not have probability greater than δ in order to hit the end node in a single walk. However, one of their direct neighboring nodes is in the target set list.

The FAST_PPR algorithm is defined in four distinct steps. The first step involves finding all the nodes close to e with a measure of the probability counted greater than $\sqrt{\delta}$ and putting them in the target set (6).

$$T = \left\{ u \in U : P(u, e) > \sqrt{\delta} \right\} \tag{6}$$

The second step involves finding the frontier set that is all the nodes satisfying the condition of being neighbor to one of the members of target set:

$$F = \left\{ w \in U \backslash T : (w, u) \in E \, for \, some \, u \in T \right\} \tag{7}$$

The third step involves finding the inverse PPR estimate from node w back to node e, with the following conditions:

$$X_i = \begin{cases} P(w, e), & W_i \; hit \; w \in F \\ 0, & W_i \; !hit \; w \in F \end{cases} \tag{8}$$

Finally, in the last step, the probability calculated from random walk is multiplied by the probability found in local update algorithm with the condition that a random walk already hit one of the nodes in frontier:

$$P(i, e) = \sum_{w \in F} P'[Walks \, from \, i, \, first \, hit \, w \in F] P''(w, e) \tag{9}$$

As result of applying these four steps, the time complexity of the algorithm is $\frac{d}{\sqrt{\delta}}$ as the upper band [17].

Approach

In this study, node reduction methods were used to reduce the size of the matrix of the graph. Then, the Monte Carlo method and local update algorithm were applied to estimate the distance and relativity from node '*i*' to node '*e*'. There could be two possible outcomes when using this methods; if the end node '*e*' could not be reached from node *i*, either; the node does not exist or the node was removed in the process of reduction. If the second conclusion is correct the map needs to be reproduced in order to spot the missing node; however, this did not occur in this study as a result of some preprocessing done prior to graphing.

As it is stated in "Terms and definitions" section, estimation over paths from *i* to *e* will involve a running time of *1/n* where *n* is the minimum number of iteration and set m contains incoming nodes to node *n.* The space needed to store all the results calculated, is equal to every node *n*'s process and their incoming nodes. However, the point is that the amount of space and time decreases as *n* decreases. Before conducting queries, some preprocessing needs to be done. At first, node reduction is applied on the starting graph so that the amount of insignificant nodes are decreased an also smaller matrixes are provided to work with. The first phase, which involves going through the PR values of all nodes in the graph and filtering the outcome, is done as an offline pre-processing; the processing time of PPR which is query based and online will not be affected by the pre-processing time complexity. The proposed reduction calculation method reduces the map to decrease the number of loops and subsets during the searching phase which occurs online during the estimation process; it is applied online as well.

Optimized relativity search (ORS) uses a new bi-directional search technique [16] to estimate the approximate time over the personalized PR in directed graphs. By applying this method, which works using the concept of model reduction, we reduce all the insignificant nodes in the graph with a given threshold are reduced, and are flagged into a new graph structure. This changes the final graph size, and number of nodes, and also divides the number of edges by a factor of *m* if and only if *m* nodes are being reduced. As a result, the number of edges will change, following the number of related nodes removed from the graph. The overall map will have fewer nodes and edges, which makes the process of searching over the graph considerably faster; this will result in changing the estimation processing time and space that is needed. The time complexity for the estimation before applying the proposed algorithm is:

$$O = \sqrt{\frac{a}{Treshold}} \tag{10}$$

where *a* is average number of edges in the graph over the number of nodes. After applying new method the time complexity will decrease to

$$O = \sqrt{\frac{a-z}{Treshold - \frac{1}{m}}} \tag{11}$$

where z is the average number of edges over the number of nodes removed from graph. And this point should be considered that the threshold given is interactive base on the overall size of the graph.

Finally, the algorithm calculates PPR estimation from initiation node i to end node e which is query based and online. The novelty in this approach lies in the fact that number of iterations happening are decreased. By removing every node in fact a chain of calculations are removed. This is because the algorithm should be iterated over each and every node. On the other hand, to reach the end point random walks that need to occur is substantially lowered, both the number of random walks and the length of each walk taken. To the best knowledge of the authors, no similar effort has been done for graph reduction in the PR estimation problem that can reduce the map without removing graph features and reducing search time to the given time complexity function.

After applying the optimizing part of the ORS algorithm, the current matrix needs to be normalized in order to gain stochastic matrix. To obtain the stochastic matrix, we need to have sums elements of every row of the matrix to sums up to one, but as not all the nodes have out degree edges its inevitable to have rows of all zero which does not sum up to one. In order to fix this problem we need to normalize the proposed matrix. In order to normalize the matrix we take the transition matrix which is:

Transition matrix:

$$P = D^{-1}A \tag{12}$$

And add 1/n to all the rows whose sum is not equal to one.

Stochastic matrix:

$$\mathrm{P}' = \mathrm{P} + \sum_{1}^{n} \frac{1}{n} A \tag{13}$$

And finally, the PR matrix will be like:

$$\mathrm{P}' = \alpha \mathrm{P}' + (1 - \alpha)\frac{I^T}{n} A \tag{14}$$

where I^{T} is an all-one matrix.

Our algorithm

In Algorithm 3, the ORS algorithm there are three main sections. First, PPR values are calculated for all the nodes in the graph. Second, a condition is defined by a certain threshold, based on the number of nodes in the graph and PPR values found. All the nodes in which do not meet the condition will be flagged in the graph. And later all the flagged nodes will be ignored by algorithm and their value won't be estimated.

Two sub-functions of Algorithm 3 are presented here. Algorithm 4 comes into play when function Set_l is called from Algorithm 3. It outputs a list of nodes close to the end node e, and estimates their closeness probability to the end node e. In Algorithm 4, the attempt is to find close neighbors to the end node. The algorithm will run over the manipulated data generated in last stage. Threshold used is square root of the given thresh hold for the whole graph, to explain the square root of threshold the PPR value

estimated in this part will be multiplied by the PPR value returned from Algorithm 5. After that, all qualified node and their relative estimated PPR value with respect to node e will be captured in a set named F_e. After that, Algorithm 5 will be read when function Set_m is called from Algorithm 3. This function is responsible for generating random walks from node i to w as well as for counting the number of walks and their estimation values.

Finally, Random Walk Estimation (Algorithm 5) defines the calculation for the second half of PPR value; this is done by calculating the number of walks from node i to w and by calculating the probability of hitting the first node in F_e. The final step, as given in algorithm 3 is to accumulate the values from Algorithms 4 and 5 if they satisfy the conditions defined.

Algorithm 3 Optimized Relativity Search (ORS) Algorithm

Input; Graph $M(V, E)$, Graph $M'(V, E)$

threshold_1 δ, threshold_r $\propto$, threshold_m γ

Output; $\grave{E}_e^i$

For $\forall v \in M$

$Pr_v = (1 - \alpha)v + \alpha$

$M_v^i = Pr_v$

If $M_v^i > \propto$

$M'_i(u) = M_v^i$

Else $M'_i(u) = 0$

Read set_l (M', δ), get $N_e, F_e, \ddot{E}_e{}^{-1}(v)$

If $i \in N^{-1}{}_e$

Calc**:** $\grave{E}_e^i$

Else

Read set_m(M', γ, N_e, F_e); get $\mathcal{M}_i(w)$, $\mathbb{C}_w^i$

Calc: $\grave{E}_e^i = \frac{1}{\mathbb{C}_w^i} \sum_{j \in \mathbb{C}_w^i} (\mathcal{M}_i(w))\ (\ddot{E}_e{}^{-1}(v))$

Algorithm 4 Close Neighbor Estimation

Input; Graph $M'(V, E)$, threshold_1 δ, tthresholder_r μ

Factor β

Output; $N_e, F_e, \ddot{\mathrm{E}}_e^{-1}(v)$

Calc: $\boldsymbol{\epsilon_{inv}} = \beta\mu$

Initialize vector $\ddot{\mathrm{E}}_e^{-1}$ and

vector $\begin{cases} \ddot{\mathrm{E}}_e^{-1}(u) = r_e(u) = 0 \; if \; u \neq e \\ \ddot{\mathrm{E}}_e^{-1}(e) = r_e(e) = \alpha \end{cases}$

Initialize set $N_e = \{e\}$ and $F_e = \{\}$

While $\exists w \in V \; and \; |w| \neq 0 \; ; \; r_e(w) > \alpha\epsilon_{inv}$

For $u \in N^{in}(w)$

$\Delta = (1-\alpha).\frac{r_e(w)}{d^{out}(u)}$

$\ddot{\mathrm{E}}_e^{-1}(u) = \ddot{\mathrm{E}}_e^{-1}(u) + \Delta$

$r_e(u) = r_e(u) + \Delta$

If $\ddot{\mathrm{E}}_e^{-1}(u) < \mu$

$F_e \leftarrow u \; and \; N_e \leftarrow u$

$r_e(u) = 0$

$F_e = F_e / N_e$

Algorithm 5 Random Walk Estimation

Input; Graph $M'(V, E)$ threshold_m γ, N_e, F_e

Output; $\mathcal{M}_i(w)$, $\mathbb{C}_w^i$

Calc: $\mathbb{C}_w^i = \frac{24 \log 10^{-6}}{\gamma}$

For $j \in \mathbb{C}_w^i$

Calc:$L_j \sim Geom(\alpha)$

Calc: Random-walk $RW_{j\,(i,L_i)}$

IF $v_j \in F_e$

Calc: v_j in set w={}

Discussion

As stated in "Related work" section, the edge-weighted PPR method is used with model reduction based on proper orthogonal decomposition (POD) and principal components analysis (PCA). Edge-weighted methods are used to calculate the PR value, whereas in ORS, a node-based searching method is used. This section describes how the edge-weighed method works and why the model reduction used in this method when it is not of any use in the optimized search model.

On reduction models based on PCA, a reduced size sample is made out of the original contribution vector. However, the contribution vector is based on edge features vector. In the proposed work, model reduction based on PCA was applied by building a reduced space out of the contribution vector of the nodes set of the graph; R is the space calculated from contribution vector $C = \{c^1, c^2, c^3, \ldots, c^r\}$, where C is the proposed edge based contribution matrix defined as:

$$C = R^C \sum \left(V^C\right)^T \tag{15}$$

As shown in (15), by applying singular value decomposition (SVD), R^C and V^C matrices are given. In the next step is to get some approximation of the reduced space U^C already made, do the calculation and make the changes to the proposed system. This means that only significant values in the contribution system will be saved as some new equations for the contribution vector, which will guide the next step on how to calculate the PPR based on reduced dimension values. As shown in (16), the reduced vector U should satisfy the condition.

$$v^T \left[A(v)\, R_y(v) - q\right] = 0 \tag{16}$$

In the last step, the most significant values are obtained from the contribution vector, and the reduced PPR vector is calculated accordingly.

$$Min \left\| v^T \left[A(v) R_y(v) - q \right] \right\|^2 s.t. \sum_j \left(R_y(v) \right)_j = 1 \tag{17}$$

For the diagonal matrix, put index I wherever the answer is 1, and ignore it if the answer is 0. However, the problem with this method is that, first, all the formulas and calculations are edge-weighted and over feature vectors as given in (17) we can make the C_w members so that their sum is equal to one using the proposed R reduce vectors accordingly. Experiments using edges, in which nodes were used in the search method, were based on the work by Xie et al. [24]. Contribution vectors in this work were calculated using:

$$q(v) = \sum_{s=1}^{m} v_s q^s \tag{18}$$

The second problem with this method is that many calculations are case-based, and depend on graphs and feature data-sets that vary. As a result, many calculation should be done in a supervised manner; however, this is time-consuming and takes a great deal of effort.

$$x^{(t+1)} = \alpha P x^t + (1 - \alpha) \tag{19}$$

Issue in calculating PR

As mentioned in "Introduction" section, the rank of a node can be calculated from ranks of nodes appointed to that node. There are three main issues with PR estimation and two groups of problems. One issue involves calculating iterative updates, and the rest involve the following. Source nodes are the nodes that have no incoming nodes, and they only have out degree nodes. Absorbing nodes are those that have only outgoing links and no incoming nodes. Cycles are the nodes following each other in succession. The problem with cycles is when they are iterated, the rank goes from one node to another without stabilizing on any fixed rank.

The issue of cycling nodes is solved in the algorithm by adding an extra condition before going through the calculation check on whether the node number is in the estimation vector: if the node id is there, ignore it; if not, go on. Issues of absorbing nodes and source nodes did not affect this study because of the normalization process conducted with the data; another factor was the jumping factor mentioned in "Related work" section, which covers any dead-end node.

Experimental results

In this experiment, the correctness of two main ideas were verified. Firstly, by applying model reduction methods, the running time of every single query was reduced by the minimum time; in addition, the larger the graph, the greater the impact of reduction. Secondly, the accuracy of results improves, as the insignificant nodes from the PPR calculation were reduced, which made the estimation of node i to e more accurate and realistic.

Set-up

This experiment was performed on a computer with Intel® Xeon(R) E5−1607 @ 3 GHz processors and 16 GB RAM. A single core was used to run the algorithm. All the matrices were loaded in the machine's main memory before calculating the time spent for the search algorithm. All the proposed algorithms were implemented in Java programming language, and the library used was Graph Implementation. A JAMA library was used for the matrices and their operations.

Data sets

The datasets used in this experiment were a Twitter data set (Directed Network), Orkut data set, and a Live Journal graph data set (Undirected Network). Details about the data sets used are given in Table 2. Because the main applications for the proposed algorithm are to estimate the distance between two nodes and to find a node by searching through neighboring nodes, social data sets are best suited for this manuscript. Tweeter and Orkut datasets used in this study were selected as the social network datasets. Similarly, the LiveJournal dataset was from a Russian social network. The Twitter, Orkut and Live Journal data sets were downloaded from a website of the Interdisciplinary Research Institute [34].

Results and discussion

The results of this experiment are twofold. The best cases were when i-th $\grave{E}_e$ value was estimated on the most popular nodes with a large number of following nodes. On average, the estimated time was cut in half as compared to previous models. The result was based on the number of tested node pairs. The lowest results occurred when the node was not a popular node; as a result, there were not many followers. Therefore, there was not much of a difference between these results and the work done in FAST_PPR for unpopular nodes; however, there still was a big difference between the results and the previously standardized methods, as shown in Fig. 1.

Figure 1a presents the lower-bound performance of the ORS algorithm as well as the local update, random surfer, and FAST_PPR algorithms. As a result of being in the lower bound, the degree of node e is not large and there is not a big difference between ORS and FAST_PPR. However, as shown in Fig. 1b, for the upper bound of the algorithm, the value for ORS are almost half that for FAST_PPR.

In order to have a clear understanding of the improvement made to the existing method when using the proposed algorithms, a comparison was made between the proposed algorithms and a previously existing bidirectional search method, FAST-PPR. The "Experimental results" section show that the proposed method worked twice faster than

Table 2 Basic information on the data sets used

Data set	Statistics	
	Num. nodes	Num. edges
Twitter	46,5017	835,423
Orkut	3,072,441	117,185,083
Live journal	4,847,571	68,993,773

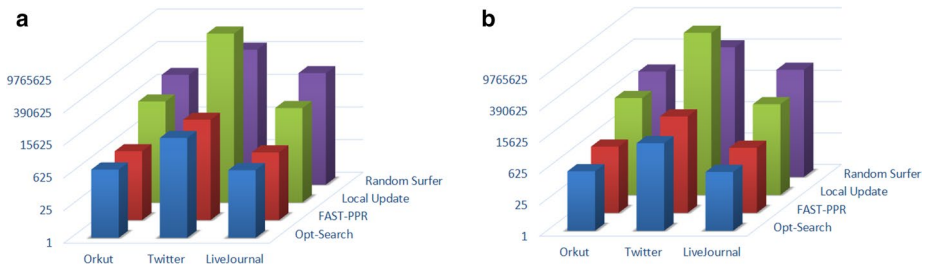

Fig. 1 **a** Minimum node reduction (*left*), **b** maximum node reduction (*right*)

FAST-PPR. For nodes having a high number of incoming degrees, ORS works in half the time and with almost the same accuracy; however, for nodes with small number of incoming degrees, the speed and accuracy was the same as when using FAST-PPR. After flagging the unpopular nodes in the adjacency matrix, they are ignored in calculating the estimation value. As a result, the final estimation value might be infected, which can result in false negative, i.e. despite existence of a connection, the algorithm return no connection. However, with further analysis, it was found out that such a challenge can be addressed with a proper threshold based on statistical properties of the nodes.

Conclusion

A new algorithm, ORS, was developed based on model reduction and a bidirectional searching method in order to optimize the processing running time of calculating PPR estimations. This algorithm estimated the importance of nodes in the graph with respect to the initiation and end nodes. Calculation was done over the optimized set of nodes in two separated steps. The first step involved going from the node i to reach node w with a predefined threshold; in addition, the estimated value was determined from node e to node u. Finally, calculations were done, as given in "Our algorithm" section, to determine if a pair counted as significant or not. The upper-bound complexity of the estimation process in large scale networks was changed from $O\left(\sqrt{\frac{a}{\delta}}\right)$ to $\left(\sqrt{\frac{a}{\delta - m}}\right)$, where a is the graph's average degree, δ is the given threshold, and m is reciprocal to the number of reduced nodes. The lower bound of running time is defined as $\Omega\left(\frac{1}{\delta}\right)$. The accuracy used to calculate probabilities is $4/n$ for the experiments. Lower bound is where there are few children for a given node and upper bound occurs when the targets node have large numbers of children.

The algorithm was tested using several measures, particularly time complexity and performance accuracy. Regarding the time complexity, the upper bound of running time changed by two factors of speed for the nodes satisfying the threshold condition. "Experimental results" section demonstrated that after applying the optimized search estimation method to the data sets, as stated in "Data sets" section, processing time improved by two factors of speed. It should be noted that these improvements apply to nodes with a high number of followers in the corresponding social networks.

Future research includes improving the PPR estimate with regard to several aspects:

1. The amount of space needed to store all the data points created by the algorithm is to be modified.

2. With the proposed method, the accuracy of the results were not as close to the accuracy of standard algorithms, which needs to be improved.
3. The lower bound of the algorithm is $\Omega\left(\frac{1}{\delta}\right)$ at the expense of a little accuracy. This can be changed to a predefined constant for all cases.
4. The ORS algorithm estimates the PPR value between a selected pair of nodes at this stage. Future research plans to expand this algorithm to all nodes in a graph.

Authors' contributions
MP, as the first author, performed the primary literature review, data collection and experiments, and also drafted the manuscript. JZ worked with MP to develop the algorithm, the paper and the framework. Both authors read and approved the final manuscript.

Authors' information
Matin Pirouz is a Ph.D. student at the Department of Computer Science, University of Nevada, Las Vegas (UNLV). Her current research work is in Big Data Analytics, Deep Learning, Network Analysis, and Graph Theory. She received a B.S. degree in Computer Science and a M.S. degree in Information Technologies from European Regional Educational Academy (EREA) in 2012 and 2014, respectively.

Dr. Justin Zhan is an associate professor at the Department of Computer Science, University of Nevada, Las Vegas (UNLV). He has previously been an associate professor at the Department of Computer Science North Carolina A&T State University. He has been a faculty member at Carnegie Mellon University and National Center for the Protection of Financial Infrastructure in Dakota State University. His research interests include Big Data, Information Assurance, Social Computing, and Health Sciences.

Acknowledgements
We gratefully acknowledge the United States Department of Defense (DoD Grants #W911NF-13-1-0130), the National Science Foundation (NSF Grant #1560625), and Oak Ridge National Laboratory (ORNL Contract #4000144962) for their support and finance for this project.

Competing interests
The authors declare that they have no competing interests.

Received: 29 March 2016 Accepted: 16 June 2016
Published online: 02 July 2016

References

1. Kamvar S, Haveliwala T, Golub G. Adaptive methods for the computation of PageRank. Linear Algebra Appl. 2004;386:51–65.
2. Haveliwala TH. Topic-sensitive PageRank: a context-sensitive ranking algorithm for web search. IEEE Trans Knowl Data Eng. 2003;15(4):784–96.
3. Brin S, Page L. The anatomy of a large-scale hypertextual web search engine. In: Proceedings of the 7th international conference on world wide web; 1998.
4. Chen P, Xie H, Maslov S, Redner S. Finding scientific gems with Google's PageRank algorithm. J Inform. 2007;1(1):8–15.
5. Hirai J, Raghavan S, Garcia-Molina H, Paepcke A. WebBase: a repository of web pages. Comput Netw. 2000;33:277–93.
6. Chakrabarti S, Dom BE, Kumar SR, Raghavan P, Rajagopalan S, Tomkins A, Gibson D, Kleinberg J. Mining the web's link structure. Computer. 1999;32(8):60–7.
7. Bharat K, Mihaila GA. When experts agree: using non-affiliated experts to rank popular topics. In: Proceedings of the 10th international conference on world wide web. New York: ACM; 2001. p. 597–602.
8. Vieira MV, Fonseca BM, Damazio R, Golgher PB, Reis DC, Ribeiro-Neto B. Efficient search ranking in social networks. In: Proceedings of the 16th ACM conference on information and knowledge management. Lisbon: ACM; 2007.
9. Yin P, Lee W-C, Lee KC. On top-K social web search. In: Proceedings of the 19th ACM international conference on information and knowledge management. New York: ACM; 2010.
10. Henzinger MR, Heydon A, Mitzenmacher M, Najork M. Measuring index quality using randomwalks on the web. Comput Netw. 1999;31(11–16):1291–303.
11. Fogaras D, Racz B. Scaling link-based similarity search. In: Proceedings of the 14th international conference on world wide web. New York: ACM Press; 2005. p. 641–50.
12. McSherry F. A uniform approach to accelerated PageRank computation. In: Proceedings of the 14th international conference on world wide web. New York: ACM Press; 2005. p. 575–82.
13. Gleich D, Polito M. Approximating personalized PageRank with minimal use of web graph data. Internet Math. 2006;3(3):257–94.
14. Weng J, Lim E-P, Jiang J, He Q. TwitterRank: finding topic-sensitive influential twitterers. In: Proceedings of the 3rd ACM international conference on web search and data mining. New York: ACM; 2010. p. 261.
15. Bahmani B, Chakrabarti K, Xin D. Fast personalized PageRank on MapReduce. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data (SIGMOD '11). New York: ACM; 2011. p. 973–84.

16. Lofgren PA, Banerjee S, Goel A, Seshadhri C. FAST-PPR: scaling personalized PageRank estimation for large graphs. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '14):, ACM, New York; 2014. p. 1436–45.
17. Andersen R, Borgs C, Chayes J, Hopcraft J, Mirrokni VS, Teng S-H. Local computation of PageRank contributions. Internet Math. 2008;5(1-2):23–45.
18. Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: bringing order to the web. Technical report. Stanford: Stanford InfoLab; 1999.
19. Haveliwala TH. Topic-sensitive PageRank. In: Proceedings of the 11th international conference on world wide web. New York: ACM Press; 2002. p. 517–26.
20. Jeh G, Widom J. Scaling personalized web search. In: Proceedings of the 12th international conference on world wide web. New York: ACM; 2003.
21. Horowitz D, Kamvar SD. The anatomy of a large-scale social search engine. In: Proceedings of the 19th international conference on world wide web. New York: ACM; 2010.
22. Fogaras D, Racz B, Csalogany K, Sarlos T. Towards scaling fully personalized PageRank: algorithms, lower bounds, and experiments. Internet Math. 2005;2(3):333.
23. Chebolu P, Melsted P. PageRank and the random surfer model. In: Proceedings of the 19th annual ACM-SIAM symposium on discrete algorithms. Society for industrial and applied mathematics; 2008. p. 1010–8.
24. Xie W, Bindel D, Demers A, Gehrke J. Edge-weighted personalized PageRank: breaking a decade-old performance barrier. In: Proceedings of the 21st ACM SIGKDD international conference on knowledge discovery and data mining (KDD '15). New York: ACM; 2015. p. 1325–34.
25. Xing W, Ghorbani AA. Weighted PageRank algorithm. In: Proceedings of the 2nd annual conference on communication networks and services research; 2004. p. 305–14.
26. Haveliwala TH. Efficient computation of PageRank. Technical report. Stanford: Stanford University; 1999.
27. Zhu F, Fang Y, Chang KCC, Ying J. Incremental and accuracyaware personalized PageRank through scheduled approximation. Proc VLDB Endow. 2013;6(6):481–92.
28. Tong H, He J, Wen Z, Lin C-Y. Diversified ranking on large graphs: an optimization viewpoint. In: Proceeding ACM SIGKDD international conference knowledge discovery data mining. New York: ACM; 2011. p. 1028–36.
29. Chakrabarti, S. Dynamic personalized PageRank in entity-relation graphs. In: Proceedings of the 16th international conference on world wide web. Banff; 2007.
30. Lofgren P, Banerjee S, Goel A. Personalized PageRank estimation and search: a bidirectional approach. Technical report; 2015.
31. Avrachenkov K, Litvak N, Nemirovsky D, Osipova N. Monte Carlo methods in PageRank computation: when one iteration is sufficient. SIAM J Numer Anal. 2007;45:890.
32. Bahmani B, Chowdhury A, Goel A. Fast incremental and personalized PageRank. Proc VLDB Endow. 2010;4:173.
33. Borgs C, Brautbar M, Chayes J, Teng S-H. Multi-scale matrix sampling and sublinear-time PageRank computation. Internet Math; 2013.
34. Interdisciplinary Research Institute, http://www.ilabsite.org/?page_id=12. Accessed 25 Dec 2015.