Personalized PageRank Solution Paths

Kyle Kloster
Purdue University
West Lafayette, IN
kkloste@purdue.edu

David F. Gleich
Purdue University
West Lafayette, IN
dgleich@purdue.edu

ABSTRACT

Personalized PageRank vectors used for many community detection and graph diffusion problems have a subtle dependence on a parameter epsilon that controls their accuracy. This parameter governs the sparsity of the solution and can be interpreted as a regularization parameter. We study algorithms to estimate the solution path as a function of the sparsity and propose two methods for this task. The first computes a full solution path and we prove it remains localized in the graph for fast runtimes. Using this method, we propose a PageRank solution path plot to diagnose new aspects of the behavior of personalized PageRank. The second method is a faster approximation to the solution path on a grid of logarithmically-spaced values that uses an interesting application of bucket sort to make the process efficient. We demonstrate that both of these algorithms are fast and local on large networks.

1. INTRODUCTION

PageRank has been used for an incredible number of applications within data mining [27] and machine learning [35], as well as the broader science community in biology [28], chemistry [25], and neuroscience [36]. (And for even more, see our recent survey [10].) Among all the uses of PageRank, the *personalized variation* is frequently used to localize the PageRank vector within a subset of the network. For instance, Voevodski et al. [31] use personalized PageRank (PPR) scores to create an affinity measure between proteins. Also, Andersen, Chung, and Lang [2] determined a relationship between PPR vectors and low-conductance sets that allowed them to create a type of graph partitioning method that does not need to see the entire graph. Their *push method* has been used for a number of important insights into communities in large social and information networks [21].

The *push method* has surprisingly complex behavior for a simple algorithm. We review it formally in Section 4.1, but we wish to start with an informal treatment. Push depends on a number of parameters: (i) a graph, (ii) a value of α for the teleportation parameter of PageRank, (iii) a set of seed nodes, and (iv) a solution tolerance of ε. The method moves PageRank mass from the seed nodes to the remainder of the graph with spreading operations that affect only the vertices neighboring previously explored vertices. It stops when the unprocessed mass at each node is less than ε scaled by the degree of that node. Consequently, given a graph, α, and seeds, we can think of the push method as producing a PPR vector that depends on ε: $\mathbf{x}_\varepsilon$.

We illustrate three PageRank vectors as we vary ε for Newman's network science collaboration graph [26] in Figure 1. There, we see that the solution vectors for PageRank[1] that result from push have only a few non-zeros for large values of ε. This is interesting because an accurate PageRank vector is mathematically non-zero everywhere in the graph. Push, with large values of ε, then produces sparse approximations to the PageRank vector. In fact, the work in a push method can be bounded *independently of the graph size*, which means that for fixed ε and α it has the same work bound for a graph with $100,000$ vertices and $100,000,000$ vertices.

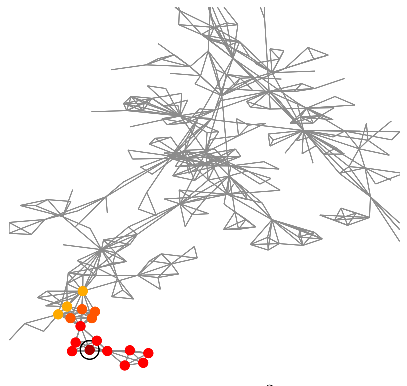
These sparse vectors, along with the speed at which they can be computed, have made the push method a frequently-used graph mining primitive. The method is typically used to identify sets of low-conductance in a graph as part of a community or cluster analysis [13, 21, 32, 12, 9]. In these cases, the size of the community or cluster returned depends on computing a vector $\mathbf{x}_\varepsilon$ and then performing a sweep-cut procedure to convert that vector into a low conductance set. Because the largest set possible is given by the non-zero elements of the vector, implementations may use up to 100 values of ε to identify communities at different size-scales [21] from given seeds.

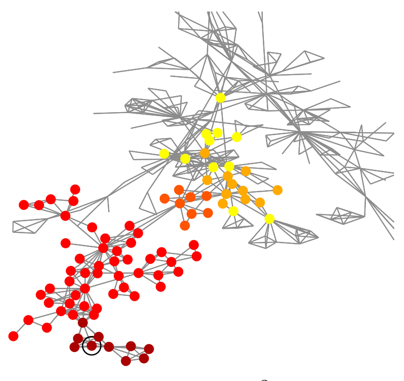
In a complementary way, we think of ε as a type of regularization parameter that controls the *sparsity* of the approximate solution. With a few additional details, this analogy is made precise [11]. We mention this connection because regularization paths and solution paths are commonly used in statistics to understand the importance and behavior of different predictors as they leave and enter optimal models. For instance, given a Lasso regularization of the least squares problem $\mathbf{Ma} \approx \mathbf{b}$,

$$\text{minimize}_{\mathbf{a}} \quad \|\mathbf{Ma} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{a}\|_1$$

[1]There is a subtle inaccuracy in this statement. As we shall see shortly, we actually are describing degree normalized PageRank values. This difference does not affect the non-zero components or the intuition behind the discussion.

KDD'15, XXX XXX XXX
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623706.


(a) $\varepsilon = 10^{-2}$


(b) $\varepsilon = 10^{-3}$

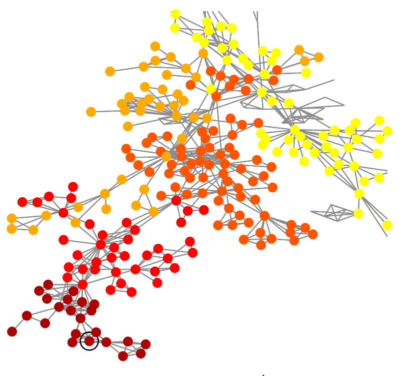

(c) $\varepsilon = 10^{-4}$

Figure 1: The color of each node reflects its value in the degree-normalized PageRank vector seeded on the circled node. The hidden nodes are mathematically zero. As the value of ε decreases, more nodes become non-zero.

the path $\mathbf{a}(\lambda)$ provides insight into the behavior of important variables in the model [30]. For example, when $\lambda = 0$, then the regularizer disappears, and the least squares solution typically uses *all of the predictors*. When λ is large, then only a sparse subset is used. For the Lasso regularizer, these solution paths can be computed in just about the same amount of work as a traditional least squares solution [8].

In this paper, we propose a set of new algorithms based on the *push procedure* that allow us to evaluate the PPR solution path as a function of ε for massive graphs. Whereas existing algorithms only work for a small number of values of ε, our algorithms show how the solution varies for *thousands of values*. In addition, we are able to compute the set of best conductance based on the sweep-cut procedure for all of these values. We use our solution path methods to explore the properties of graphs in the next section. In our technical description, we show that our method remain localized in the graph. It has a runtime that is quadratic with respect to the push procedure—and in real networks with millions of nodes it runs in less than a second.

Summary of contributions

- We present a method for studying the *solution paths* of personalized PageRank diffusion vectors, and use it to investigate both small and large networks (Section 2).
- We detail a specific type of *PageRank solution path plot* that reveals important information about the behavior of the solutions as ε varies, as well as the small conductance sets identified by the algorithm.
- We prove that our personalized PageRank path computation is independent of the size of the graph (Theorem 1).
- For applications such as network community profiles, we develop an efficient approximation to the solution path at a prescribed, logarithmically-spaced grid of ε values. (This could be considered an *optimized* version of what was done before.) This method can compute solutions for hundreds of values of ε in time that is roughly twice that of optimized codes for the independent runs of the push method.
- We also evaluate our methods on a set of large graphs and show that they find sets of slightly better conductance than prior methods. This has important implications for community detection methods that use this as a subroutine.

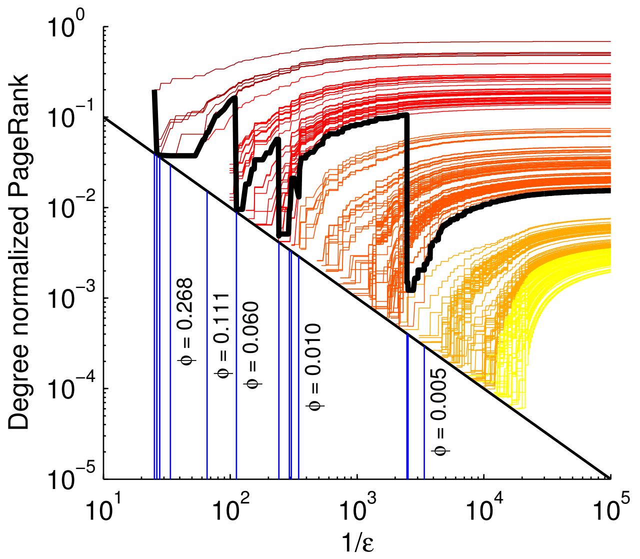


Figure 2: A degree-normalized PPR solution path plot for the network science dataset using the same seed node as Figure 1. Each curve is the path of a single entry of the solution $\mathbf{x}_\varepsilon$. The solution paths are colored based on the PageRank values at $\varepsilon = 10^{-4}$. This reflects around 21,000 values of ε. The dark black line shows the threshold where we find the set of best conductance from a sweep cut.

We plan to make our experimental codes available in the spirit of reproducible research.

2. PERSONALIZED PAGERANK PATHS

We now show the types of insights that our solution path methodology can provide. We should illustrate that these are primarily designed for human interpretation. Our vision is that they would be used by an analyst that was studying a network and needed to better understand the "region" around a target node. These solution paths would then be combined with something like a graph layout framework to highlight these patterns in the graph.

PageRank solution path plots. We introduce *degree normalized PageRank solution path plots* to reveal informative structure inside the solution paths. An example we'll discuss shortly is Figure 2. For each graph vertex represented in the degree normalized PageRank vector $\mathbf{x}_\varepsilon$, we show its solution trajectory as we vary ε through all values of ε output from our solution path procedure. The color of each line is given by the solution value at a particular value of ε (such as 10^{-4} in that plot.) We display these on a log-log scale where the horizontal axis is $1/\varepsilon$. This means that *sparse vectors* are at the left and we introduce nodes as we move from left to right. Due to details of the push method, the minimum PageRank value for a solution with tolerance ε is ε; hence we show the black diagonal line for these points. As nodes enter the solution, they move or jump off of this line. The vertical blue lines in the bottom left of the plot show the values of ε where we detect a new set of best conductance. Representative conductance values are shown when there is room in the plot. The dark black line in the upper triangle shows the threshold that determines the set of best conductance at a given value of ε. This allows us to follow the minimum conductance set trajectory as we vary ε.

Network science collaborators. In Figure 2, we show the PageRank solution path for around $21,000$ values of ε computed via our algorithm for the network science collaboration network. This computation runs in less than a second. Here, we see that large gaps in the degree normalized PageRank vector indicate cutoffs for sets of high conductance. This behavior is known to occur when sets of really good conductance emerge [1]. We can now see how they evolve and how the procedure quickly jumps between them.

Facebook. On a crawl of a Facebook network from 2009 where edges between nodes correspond to observed interactions [33] (see Table 1, `fb-one-cc`, for the statistics), we are able to find a large, low conductance set using our solution path method. (Again, this takes about a second of computation.) This diffusion shows no sharp drops in the PageRank values like in the network science data. Yet we still find good conductance cuts. Note the few stray "orange" nodes in the sea of yellow. These nodes quickly grow in PageRank and break into the set of smallest conductance. Finding these nodes is likely to be important to understand the boundaries of communities in social networks; these trajectories could also indicate anomalous nodes.

YouTube communities. Finally, we use our tools to study a community detection problem as posed by Kloumann and Kleinberg [18]. They found that using PageRank values, instead of degree normalized PageRank values, produced better results in a seeded community detection problem. In Figure 4 we show the solution paths for the YouTube graph [24], seeded with 10% of the nodes from a single ground-truth community, for both the degree normalized PageRank vector we will usually use, as well as the regular PageRank vector. (For the PageRank vector, we simply scale our solution.) We show this both for the seeds of the community as well as the remaining nodes. These plots illustrate that understanding how the seeds behave is critical to appreciating the diffusion. In the unnormalized version, the seeds lift the values of a large number of nodes, whereas these are scattered throughout the vector in the degree normalized version.

Additional opportunites. Fast access to the solution path trajectories provides a number of additional opportunties that we have not yet explored. For instance, nodes in

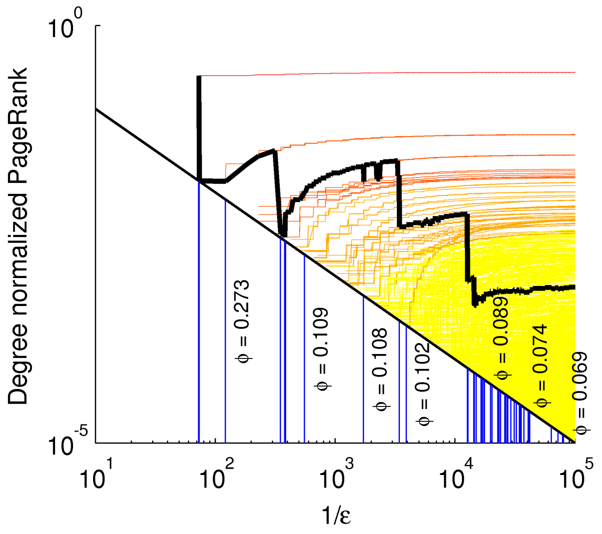


Figure 3: The degree-normalized PageRank solution path for a crawl of observed Facebook network activity for one year shows large, good cuts do not need to have drops in the PageRank vectors. Nodes enter the solution and then quickly break into the best conductance set, showing that the frontier of the diffusion should be an interesting set in this graph.

an egonet could be clustered by properties of their solution paths instead of their connectivity patterns. These trajectories could also be new features to role discovery frameworks such as RolX [15].

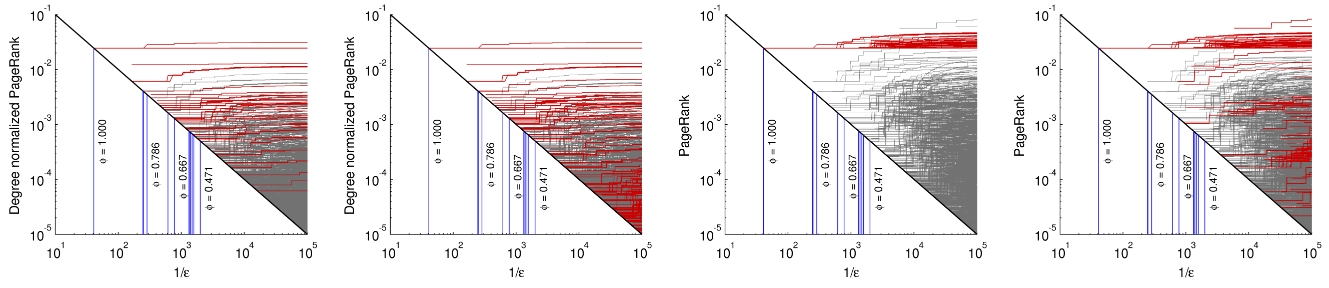
3. TECHNICAL PRELIMINARIES

Before presenting our method for computing these solution paths, we first fix our notation. We denote a graph by $G = (V, E)$, where V is the set of nodes and E the set of edges. All graphs we consider are simple and undirected. Let G have $n = |V|$ nodes and fix a labeling of the graph nodes using the numbers 1, 2, $\ldots$, n. We refer to a node by its label. For each node j we denote its degree by d_j.

Random walk transition matrix. We analyze PageRank from the perspective of matrix computations. We define the *adjacency matrix* of the graph G, which we denote by $\mathbf{A}$, to be the $n \times n$ matrix having $A_{i,j} = 1$ if nodes i and j are connected by an edge, and 0 otherwise. Since G is simple and undirected, $\mathbf{A}$ is symmetric with 0s on the diagonal. Then the matrix $\mathbf{D}$ denotes the diagonal matrix with entry (i, i) equal to the degree of node i, d_i. Assuming G has no isolated nodes, $\mathbf{D}$ is invertible, and we can define the *random walk transition matrix* $\mathbf{P} := \mathbf{A}\mathbf{D}^{-1}$. Lastly, we denote by $\mathbf{e}_j$ the standard basis vector of appropriate dimensions with a 1 in entry j, and by $\mathbf{e}$ the vector of all 1s.

In general, we use subscripts on matrices and vectors to denote entries, e.g. $A_{i,j}$ is entry (i, j) of matrix $\mathbf{A}$; the notation for standard basis vectors, $\mathbf{e}_j$, is an exception. Superscripts refer to vectors in a sequence of vectors, e.g. $\mathbf{x}^{(k)}$ is the kth vector in a sequence.

Conductance. For any set of nodes, $S \subseteq V$, we define the *volume* of S to be the sum of the degrees of the nodes in S, denoted $\text{vol}(S) = \sum_{j \in S} d_j$. Next, define the *boundary* of



(a) Seed trajectories (b) Community trajectories (c) Seed trajectories (d) Community trajectories

Figure 4: Seed and ground-truth community trajectories in degree normalized PageRank vectors and unnormalized PageRank vectors show that both the seed nodes and community nodes remain repressed in the sorted degree normalized vector but are raised in the unnormalized trajectories. This illustrates a finding from Kloumann and Kleinberg [18].

$S \subseteq V$ to be the set of edges that have one endpoint inside S and the other endpoint outside S, denoted $\partial(S)$. Finally, the *conductance* of S, denoted $\phi(S)$, is defined by

$$\phi(S) := \frac{|\partial(S)|}{\min\{\text{vol}(S), \text{vol}(V - S)\}}.$$

Intuitively, the conductance of S is the probability of leaving S if you take a random walk of length one starting at a node chosen uniformly at random inside S. Conductance can be thought of as measuring the extent to which a set is more connected to itself than the rest of the graph. Because of this, it is widely used as a metric in searching for communities focused around a small set of starting nodes. We call such starting nodes *seed sets* and the resulting communities, *local* communities. A common means of identifying good-conductance sets is via a graph diffusion. We focus on the most commonly used diffusion, the PageRank diffusion, which we now review.

PageRank Objective. For a stochastic matrix $\mathbf{P}$, a stochastic vector $\mathbf{v}$, and a parameter $\alpha \in (0, 1)$ we define the PageRank graph diffusion as the solution $\mathbf{x}$ to the linear system

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}. \tag{1}$$

When $\mathbf{v} = (1/|S|)\mathbf{e}_S$ — i.e. the indicator vector for a seed set S, normalized to be stochastic — then we say the PageRank vector has been *personalized* to the set S.

Given PageRank diffusion scores $\mathbf{x}_j$, the values $\mathbf{x}_j/d_j$ determine node rankings, and a sweep-cut procedure (described below) determines a set of good conductance. We would like to bound the error in approximating the values $\mathbf{x}_j/d_j$, then. In particular, we want to guarantee that our approximate solution $\hat{\mathbf{x}}$ gives scores $\hat{\mathbf{x}}_j$ that satisfy $0 \leq \mathbf{x}_j - \hat{\mathbf{x}}_j < \varepsilon d_j$. In vector notation, we want to ensure that

$$\mathbf{x} \geq \hat{\mathbf{x}}, \quad \text{and} \quad \|\mathbf{D}^{-1}\mathbf{x} - \mathbf{D}^{-1}\hat{\mathbf{x}}\|_\infty < \varepsilon. \tag{2}$$

In Section 4.2 we provide a convergence criterion for our algorithms that guarantees this condition holds.

Sweep-cut procedure. Once a PPR diffusion $\mathbf{x}$ is computed, its best-conductance set is produced with a *sweep cut* procedure, described as follows. Rank the nodes in descending order by their scaled diffusion scores $\mathbf{x}_j/d_j$, with large scores ranking the highest. We denote the set of nodes ranked 1 through m by $S(m)$. Once the nodes are ranked, we iteratively compute the conductance of the sets $S(m)$ for $m = 2, 3, \ldots, \text{nonzeros}(\hat{\mathbf{x}})$. The set $S(t)$ with the best conductance is then output as the set of best conductance for that diffusion. We use these preliminary procedures in our algorithms for approximating PPR diffusions.

4. ALGORITHMS

Here we present two novel algorithms for analyzing a PPR diffusion across a variety of accuracy parameter settings by computing the diffusion only a single time. Our first algorithm computes the best-conductance set from PPR diffusions for *every* accuracy satisfied in an interval $[\varepsilon, \varepsilon_{\max}]$, where ε and $\varepsilon_{\max}$ are inputs. We prove the total runtime is bounded by $O(\varepsilon^{-2}(1 - \alpha)^{-2})$, though we believe improvements can be made. In addition to identifying the best-conductance set taken from the different approximations, the algorithm enables us to study the solution paths of PageRank, i.e. how the PPR diffusion scores change as the diffusion's accuracy varies. Hence, we call this method `ppr-path`.

We describe a second algorithm optimized for speed, as the exhaustive nature of our first method generates too much intermediate data for stricter values of ε. Instead of computing the full solution paths, the second method searches for good-conductance sets over an approximate solution for each accuracy parameter taken from a grid of parameter values. The spacing of the accuracy parameters values on the grid is an additional input parameter. For this reason, we call the algorithm `ppr-grid`. For a log-spaced grid of values $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_N$ with strictest accuracy ε, we locate the best-conductance set taken from a sweep over each ε_k-approximation. The work required to compute the diffusions is bounded by $O(\varepsilon^{-1}(1-\alpha)^{-1})$; we show this yields a constant factor speedup over the practice of computing each diffusion separately. However, our method requires the same amount of work for performing the sweeps over each different diffusion.

Both of our methods take their fundamental operation from a sparse iterative solver similar to the Gauss-Seidel coordinate relaxation scheme. This operation is the basis for a number of graph algorithms, including the constant time method presented in [2]. We review the operation here in the language of matrix computations, and provide a convergence criterion that we prove guarantees the desired accuracy condition (2).

4.1 Push operation

The iterative solver we present here is applicable outside our narrow setting, and so we present the scheme in full generality before recasting it for PageRank.

Let $\mathbf{Mx} = \mathbf{b}$ be an arbitrary, square linear system. Let $\mathbf{x}^{(k)} \approx \mathbf{x}$ be an iterative solution after k steps. Then the corresponding residual is $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Mx}^{(k)}$. Let j be a row index where we want to *relax* the equation, and let r be the residual value there, $r = \mathbf{r}_j^{(k)}$. We update the solution and residual as follows: add r to the corresponding entry of the solution vector, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + r\mathbf{e}_j$. Then, compute $\mathbf{r}^{(k+1)}$ from the definition of the residual, $\mathbf{r}^{(k+1)} = \mathbf{b} - \mathbf{Mx}^{(k+1)}$. Expanding $\mathbf{x}^{(k+1)}$ in this yields, after some light algebra,

$$\begin{aligned}\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + r\mathbf{e}_j\\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - r\mathbf{Me}_j.\end{aligned}$$

Note that the update requires updating just one entry of $\mathbf{x}^{(k)}$ and accessing only a single column of the matrix $\mathbf{M}$.

A modified linear system. Rather than apply the scheme to the PageRank linear system, it is easier for our purposes to use a modified system. We multiply Equation (1) by $\mathbf{D}^{-1}$ to obtain, after some manipulation,

$$(\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{D}^{-1}\mathbf{x} = (1-\alpha)\mathbf{D}^{-1}\mathbf{v}.$$

Note this transformation relies on $\mathbf{A}$ being symmetric so that $\mathbf{P}^T = (\mathbf{AD}^{-1})^T = \mathbf{D}^{-1}\mathbf{A} = \mathbf{D}^{-1}\mathbf{PD}$. To avoid writing $\mathbf{D}^{-1}$ repeatedly, we make the change of variables $\mathbf{y} = (1/(1-\alpha))\mathbf{D}^{-1}\mathbf{x}$ and $\mathbf{b} = \mathbf{D}^{-1}\mathbf{v}$. The modified system is then

$$(\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{y} = \mathbf{b}, \qquad \mathbf{r}^{(k)} = \mathbf{b} - (\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{y}^{(k)}, \quad (3)$$

and we set $\mathbf{x}^{(k)} = (1-\alpha)\mathbf{Dy}^{(k)}$. Substituting this system into the iterative solver above and simplifying yields the iterative update

$$\begin{aligned}\mathbf{y}^{(k+1)} &= \mathbf{y}^{(k)} + r\mathbf{e}_j\\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - r\mathbf{e}_j + r\alpha\mathbf{P}^T\mathbf{e}_j. \quad (4)\end{aligned}$$

This residual update corresponds to deleting a selected entry j, and then for each neighbor i of node j, adding $(r\alpha)/d_i$ to entry i of the residual. An important consequence of this, which we use below, is that the residual is always nonnegative.

4.2 Convergence criterion

Now we give a criterion for convergence of the iteration (4) and show that the resulting vector $\hat{\mathbf{y}}$ yields a PPR approximation $\hat{\mathbf{x}}$ satisfying the desired accuracy (2).

The error in the system (3) can be expressed in terms of the residual vector. Left-multiplying the residual in (3) by $(\mathbf{I} - \alpha\mathbf{P}^T)^{-1}$ and substituting $\mathbf{y} = (\mathbf{I} - \alpha\mathbf{P}^T)^{-1}\mathbf{b}$, we get

$$\mathbf{y} - \mathbf{y}^{(k)} = \left(\sum_{m=0}^{\infty} \alpha^m \left(\mathbf{P}^T\right)^m\right)\mathbf{r}^{(k)},$$

where the right-hand side replaces $(\mathbf{I} - \alpha\mathbf{P}^T)^{-1}$ with its Neumann series. Note here that, because the right-hand side consists of all nonnegative entries, we must have $\mathbf{y} - \mathbf{y}^{(k)} \geq 0$. Since $(1-\alpha)\mathbf{y}^{(k)} = \mathbf{D}^{-1}\mathbf{x}^{(k)}$, this implies $\mathbf{x} \geq \mathbf{x}^{(k)}$, proving one component of the accuracy criterion (2) is satisfied.

Using the triangle inequality and sub-multiplicativity of the infinity norm allows us to bound $\|\mathbf{y} - \mathbf{y}^{(k)}\|_\infty$ with

$$\sum_{m=0}^{\infty} \alpha^m \left\|\left(\mathbf{P}^T\right)^m \mathbf{r}^{(k)}\right\|_\infty \leq \left(\sum_{m=0}^{\infty} \alpha^m \left\|\mathbf{P}^T\right\|_\infty^m\right)\left\|\mathbf{r}^{(k)}\right\|_\infty.$$

Finally, since $\mathbf{P}$ is column stochastic, $\mathbf{P}^T$ is row-stochastic, and so $\|\mathbf{P}^T\|_\infty = 1$. Substituting this and noting that $\sum_{m=0}^{\infty} \alpha^m = 1/(1-\alpha)$ allows us to bound

$$\tfrac{1}{1-\alpha}\left\|\mathbf{D}^{-1}\mathbf{x} - \mathbf{D}^{-1}\mathbf{x}^{(k)}\right\|_\infty = \left\|\mathbf{y} - \mathbf{y}^{(k)}\right\|_\infty \leq \tfrac{1}{1-\alpha}\left\|\mathbf{r}^{(k)}\right\|_\infty.$$

Then iterating until reaching the convergence criterion

$$\left\|\mathbf{r}^{(k)}\right\|_\infty < \varepsilon \quad (5)$$

will guarantee that the accuracy condition (2) is obtained. This corresponds to computing $\hat{\mathbf{y}}$ satisfying $\|\mathbf{y} - \hat{\mathbf{y}}\|_\infty < \frac{\varepsilon}{1-\alpha}$.

A choice of accuracy ε gives rise to an infinite number of approximations $\mathbf{x}_\varepsilon$ satisfying our accuracy criteria (2). Moreover, operating on residual entries in different orders can result in different vectors $\mathbf{x}_\varepsilon$ for the same ε parameter setting. To introduce some regularity to this, at each step of our method `ppr-path`, we operate on the largest entry of the residual. Even this is not totally deterministic, as multiple residual entries can have the exact same value at the same step, which then requires deciding on one entry.

We can achieve uniqueness of each approximation $\mathbf{x}_\varepsilon$ using a particular form of 1-norm regularization [11]. However, this requires a more intricate implementation, and it is not clear that the uniqueness attained would provide any meaningful change. We believe the informal "uniqueness" achieved by using the maximum residual entry provides nearly identical, if not exactly identical, results. Next we lay out a systematic way for computing a PPR diffusion for each possible accuracy ε using this iterative update.

4.3 PageRank solution paths

Recall our goal for computing the solution path for all ε values. Let $\mathbf{P}$ be a stochastic matrix, choose α satisfying $0 < \alpha < 1$, let $\mathbf{v}$ be a stochastic vector, and set $\mathbf{b} = \mathbf{D}^{-1}\mathbf{v}$. Fix input parameters ε and $\varepsilon_{\max}$. Then for each value $\varepsilon_{\text{cur}} \in [\varepsilon, \varepsilon_{\max}]$, we want an approximation of the solution to $(\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{y} = \mathbf{b}$ that satisfies $\|\mathbf{y} - \hat{\mathbf{y}}\|_\infty < \frac{\varepsilon_{\text{cur}}}{1-\alpha}$.

Given initial solution $\mathbf{y}^{(0)} = \mathbf{0}$ and residual $\mathbf{r}^{(0)} = \mathbf{b}$, proceed as follows. Maintain a priority queue, $Q(\mathbf{r})$, of all entries of the residual that do not satisfy the convergence criterion $\mathbf{r}_j < \varepsilon$. We store the entries of $Q(\mathbf{r})$ using a max-heap so that we can quickly determine $\|\mathbf{r}\|_\infty$ at every step.

Each time the value $\|\mathbf{r}\|_\infty$ reaches a new minimum, we consider the resulting solution vector to satisfy a new "current" accuracy, which we denote ε_{cur}. For each such ε_{cur} achieved, we want to perform a sweep over the solution vector. Because the sweep operation requires a sorted solution vector, we keep $\mathbf{y}$ in a sorted array, $L(\mathbf{y})$. By re-sorting the solution vector each time a single entry $\mathbf{y}_j$ is updated, we avoid having to do a full sweep for each "new" ε_{cur}-approximation. The local sorting operation is a bubblesort on a single entry; the local sweep update we describe below.

With the residual and solution vector organized in this way, we can quickly perform each step of the above iterative update. Then, iterating until $\|\mathbf{r}\|_\infty < \varepsilon$ guarantees convergence to the desired accuracy. Next we present the iteration in full detail.

PPR path algorithm. The `ppr-path` algorithm performs the following iteration until the maximum entry in $Q(\mathbf{r})$ is below the smallest parameter desired, ε.

1. Pop the max of $Q(\mathbf{r})$, say entry j with value r, then set $\mathbf{r}_j = 0$ and reheap $Q(\mathbf{r})$.
2. Add r to $\mathbf{y}_j$.

   (a) Bubblesort entry $\mathbf{y}_j$ in $L(\mathbf{y})$.
   (b) If $L(\mathbf{y})$ changes, perform a local sweep update.
3. Add $r\alpha\mathbf{P}^T\mathbf{e}_j$ to $\mathbf{r}$.
4. For each entry i of $\mathbf{r}$ that was updated, if it does not satisfy $r_i < \varepsilon$, then insert (or update) that entry in $Q(\mathbf{r})$ and re-heap.
5. If $\|\mathbf{r}\|_\infty < \varepsilon_{\text{cur}}$, record the sweep information, then set $\varepsilon_{\text{cur}} = \|\mathbf{r}\|_\infty$.

When the max-heap $Q(\mathbf{r})$ is empty, this signals that all entries of $\mathbf{r}$ satisfy the convergence criterion $r_j < \varepsilon$, and so our diffusion score approximations satisfy the accuracy requirement (2).

Sweep update. The standard sweep operation over a solution vector involves sorting the entire solution vector and iteratively computing the conductance of each consecutive sweep set. Here, we re-sort the solution vector after each update by making only the local changes necessary to move entry $\mathbf{y}_j$ to the correct ranking in $L(\mathbf{y})$. This is accomplished by bubblesorting the updated entry $\mathbf{y}_j$ up the rankings in $L(\mathbf{y})$. Note that if $\mathbf{y}^{(k)}$ has T_k nonzero entries, then this step can take at most T_k operations. We believe this loose upperbound can be improved. We *could* determine the new rank of node $\mathbf{y}_j$ in work $\log T_k$ via a binary insert. However, since we must update the rank and sweep information of each node that node $\mathbf{y}_j$ surpasses, the asymptotic complexity would not change.

Once the node ranks have been corrected, the conductance score update proceeds as follows. Denote by $S^{(k-1)}(m)$ the set of nodes that have rankings $1, 2, \cdots, m$ during step $k-1$. Assuming we have the cut-set (cut and volume) information for each of these sets, then we can update that information for the sets $S^{(k)}(m)$ as follows.

Suppose the node that changed rankings was promoted from rank j to rank $j - \Delta_k$. Observe that the sets $S^{(k)}(m)$ and their cut-set information remain the same for any set $S^{(k)}(m)$ lying inside the rankings $[1, \cdots, j-\Delta_k - 1]$, because the change in rankings happened entirely in the interval $[j-\Delta_k, \cdots, j]$. This occurs for $m < j - \Delta_k$. Similarly, any set $S^{(k)}(m)$ with $m > j$ would already contain all of the nodes whose rank changed – altering the ordering within the set does not alter the conductance of that set, and so this cut-set information also need not be changed. Hence, we need to update the cut-set information for only the intermediate sets.

Now we update the cut-set information for those intermediate sets. We refer to the node that changed rank as node $L(j)$. Its old rank was j, and its new rank is $j - \Delta_k$. Note that the cut-set information for the set $S^{(k)}(j-t)$ (for $t = 0, \cdots, \Delta_k$) is the exact same as that of set $S^{(k-1)}(j-t-1) \cup \{L(j)\}$. In words, we introduce the node $L(j)$ to the set $S^{(k-1)}(j-t-1)$ from the previous iteration, and then compute the cut-set information for the new iteration's set, $S^{(k)}(j-t)$, by looking at just the neighborhood of node $L(j)$ a single time. This provides a great savings over simply reperforming the sweep procedure over the entire solution vector up to the index where the rankings changed.

If the node being operated on, $L(j)$, has degree d, then this process requires work $O(d + \Delta_k)$. As discussed above, we can upperbound Δ_k with the total number of iterations the algorithm performs T_k.

THEOREM 1. *Given a random walk transition matrix $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{D}^{-1}$, stochastic vector $\boldsymbol{v}$, and input parameters $\alpha \in (0,1)$ and $\varepsilon_{\max} > \varepsilon > 0$, our* `ppr-path` *algorithm outputs the best-conductance set found from sweeps over ε_{cur}-accurate degree-normalized solution vectors $\hat{\boldsymbol{x}}$ to $(\boldsymbol{I} - \alpha\boldsymbol{P})\boldsymbol{x} = (1-\alpha)\boldsymbol{v}$, for all values $\varepsilon_{cur} \in [\varepsilon, \varepsilon_{\max}]$. The total work required is bounded by $O\left(\frac{1}{\varepsilon^2(1-\alpha)^2}\right)$.*

Proof. We carry out the proof in two stages. First, we show that the basic iterative update converges in work $O(\varepsilon^{-1}(1-\alpha)^{-1})$. Then, we show that the additional work of sorting the solution vector and sweeping is bounded by $O(\varepsilon^{-2}(1-\alpha)^{-2})$.

Push work. We count the work on just the residual $\mathbf{r}^{(k)}$ and solution vector $\mathbf{y}^{(k)}$. The work required to maintain the heap Q and sorted array L is accounted for below.

Each step, the push operation acts on a single entry in the residual that satisfies $r_j \geq \varepsilon$. The step consists of a constant number of operations to update the residual and solution vectors (namely, updating a single entry in each), then adding the constant $(r\alpha)/d_j$ to $\mathbf{r}_i^{(k)}$ for each neighbor i of node j. Since j has d_j such neighbors, the total work in one step is bounded by $O(d_j)$.

If T steps of the push operation are performed, then the amount of work required to obtain an accuracy of ε is bounded by $\sum_{t=0}^{T} d_j$, where $j = j(t)$ is the index of the residual operated on in step t, $\mathbf{r}_j^{(t)}$.

Next we bound this expression for the work done in these "push" steps. Since all entries of the solution and residual vectors are nonnegative at all times, the sum of the values r_t pushed at each step exactly equals the sum of the values $\mathbf{y}^{(k)}$, i.e. $\sum_{t=0}^{T} r_t = \mathbf{e}^T\mathbf{y}^{(k)}$. Since $\mathbf{y}^{(k)} = (1/(1-\alpha))\mathbf{D}^{-1}\mathbf{x}^{(k)}$, we then have that the sum of entries in $(1/(1-\alpha))\mathbf{x}^{(k)}$ equals the sum of values pushed from the residual scaled by degree and $(1-\alpha)$, i.e. $\mathbf{e}^T\mathbf{x}^{(k)} = (1-\alpha)\sum_{t=0}^{T} r_t \cdot d_{j(t)}$, where $j(t)$ is the node pushed in step t. We claim that the sum $\mathbf{e}^T\mathbf{x}^{(k)} \leq 1$. Assuming this for the moment, we get from the previous equation that $(1-\alpha)\sum_{t=0}^{T} r_t \cdot d_{j(t)} = \mathbf{e}^T\mathbf{x}^{(k)} \leq 1$. But each step of `ppr-path` operates on a residual value satisfying $r_t \geq \varepsilon$, so we have

$$(1-\alpha)\sum_{t=0}^{T} \varepsilon \cdot d_{j(t)} < (1-\alpha)\sum_{t=0}^{T} r_t \cdot d_{j(t)} \leq 1.$$

Dividing by $\varepsilon(1-\alpha)$ completes the proof that the expression for work, $\sum_{t=0}^{T} d_{j(t)}$, is bounded by $O\left(\varepsilon^{-1}(1-\alpha)^{-1}\right)$.

Lastly, we justify the claim $\mathbf{e}^T\mathbf{x}^{(k)} \leq 1$. Left-multiplying the equations in (3) by $(\mathbf{D}\mathbf{e})^T$ and using stochasticity of $\mathbf{v}$ gives

$$\begin{aligned} \mathbf{e}^T(\mathbf{I} - \alpha\mathbf{P})\mathbf{D}\mathbf{y}^{(k)} &= \mathbf{e}^T\mathbf{D}\mathbf{b} - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)} \\ (1-\alpha)\mathbf{e}^T\tfrac{1}{(1-\alpha)}\mathbf{x}^{(k)} &= \mathbf{e}^T\mathbf{v} - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)} \\ \mathbf{e}^T\mathbf{x}^{(k)} &= 1 - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)}. \end{aligned} \quad (6)$$

As noted above, all entries of the residual and iterative solution vector are nonnegative at all times. The sum $\mathbf{e}^T\mathbf{x}^{(k)}$ cannot exceed 1, then, because that would imply that the residual summed to a negative number, contradicting nonnegativity of the residual vector. Hence, $\mathbf{e}^T\mathbf{x}^{(k)} \leq 1$.

Sorting and sweeping work. Here we account for the work performed each step in maintaining the residual heap $Q(\mathbf{r})$, re-sorting the solution vector $L(\mathbf{y})$, and updating the sweep information for $L(\mathbf{y})$. To ease the process, we first fix some notation: denote the number of entries in the residual heap $Q(\mathbf{r})$ by $|Q|$, and the number of non-zero entries in the sorted solution vector $L(\mathbf{y})$ by $|L|$. We will bound both of these quantities later on. We continue to use Δ_t to denote the number of rank positions changed in $L(\mathbf{y})$ in step t. Finally, recall that T denotes the number of iterations of the algorithm required to terminate.

The work bounds we will prove, listed in the order in which the `ppr-path` algorithm performs them, are as follows:

Operation	actual work	upperbound
Find $\max(\mathbf{r})$	1	1
Delete $\max(\mathbf{r})$	$\log(\|Q\|)$	$\log(\frac{1}{\varepsilon(1-\alpha)})$
Bubblesort $L(\mathbf{y}_j)$	Δ_t	T
Re-sweep $L(\mathbf{y})$	$d_j + \Delta_t$	$d_j + T$
Update $\mathbf{r} + r\alpha\mathbf{P}^T\mathbf{e}_j$	d_j	d_j
Re-heap $Q(\mathbf{r})$	$d_j \log(\|Q\|)$	$d_j \log(\frac{1}{\varepsilon(1-\alpha)})$

The residual heap operations for deleting max $Q(\mathbf{r})$, and re-heaping the updated entries each require $O(\log(|Q|))$ work, where $|Q|$ is the size of the heap, i.e. the number of nonzero entries in the residual. We can upperbound this number using the total number of pushes performed (since a nonzero in the residual can exist only via a push operation placing it there). We bound $|Q|$ by $O(\varepsilon^{-1}(1-\alpha)^{-1})$, then.

Re-sorting the solution vector via a bubblesort can involve no more operations than the length of the solution vector. Since a nonzero in entry $\mathbf{y}_j$ can exist only if a step of the algorithm operates on an entry $\mathbf{r}_j$, the number of nonzeros in $\mathbf{y}$ is bounded by the number of steps of the algorithm, i.e. $|L| \leq T$. We believe this bound to be loose, but cannot currently tighten it. Note that the work required in updating sweep information also requires Δ_t work, which we again upperbound by T. The d_j term in updating sweep information is from accessing the neighbors of the entry $\mathbf{y}_j$, the node changing its rank.

The dominant terms in the above expression for work are the re-heap updates and the bubblesort and re-sweep operations, which require a total of $O(d_j \log(|Q|) + |L|)$ work each step. Summing this over all T steps of the algorithm, we can majorize work by $O(\log(|Q|) \cdot \sum_{t=0}^{T} d_j) + O(\sum_{t=0}^{T} |L|)$, which is upperbounded by $O\left(\frac{1}{\varepsilon(1-\alpha)} \log(|Q|) + T \cdot |L|\right)$. Finally, substituting in our loose upperbounds for T, $|Q|$, and $|L|$ mentioned above completes the proof:

$$O\left(\frac{1}{\varepsilon(1-\alpha)} \log(\frac{1}{\varepsilon(1-\alpha)}) + \frac{1}{\varepsilon^2(1-\alpha)^2}\right) \leq O\left(\frac{1}{\varepsilon^2(1-\alpha)^2}\right).$$

4.4 Fast multi-parameter PPR

Here we present a fast framework for computing ε-approximations of a PPR diffusion without computing a new diffusion for each ε. This enables us to identify the optimal output that would result from multiple diffusion computations for different ε values, but without having to do the work of computing a new diffusion for each different ε.

The framework is compatible with every set of parameter choices that allows for constant-time bin look-ups. More precisely, the set of parameters $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_N$ must have an efficient method for determining the index k such that, given a value r, we have $\varepsilon_{k-1} > r \geq \varepsilon_k$. We focus on a set of ε values that are taken from a log-spaced grid: that is, the parameters are of the form $\varepsilon_k = \varepsilon_0\theta^k$ for constants $0 < \varepsilon_0, \theta < 1$. Because we assume our ε parameters are taken from such a grid, we call our method `ppr-grid`. Another possibly useful case is choosing ε_k values taken from a Chebyshev-node like grid, allowing for constant-time shelf-placement via $\cos^{-1}$ evaluations.

We emphasize that the underlying algorithm we use to compute the PageRank diffusion is closely related to the the push method discussed in Section 4.1 as implemented by [2]; in the case that only a single accuracy parameter is used, the algorithms are identical. When more than one accuracy setting is used, we employ a special data structure, which we describe next.

Shelf structure.

The main difference between our algorithm `ppr-grid` and previous implementations of the push method lies in our data structure replacing the priority queue, Q, discussed in `ppr-path`. Instead of inserting residual entries in a heap as in `ppr-path`, we organize them in a system of arrays. Each array holds entries between consecutive values of ε_k, so that each array holds entries larger than the shelf below it. For this reason, we call this system of arrays a "max-shelf", H, and refer to each individual array as a "shelf", H_k.

The process is effectively a bucket sort: each shelf (or bucket) of H holds entries of the residual lying between consecutive values of ε_k in the parameter grid. For parameters $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_N$, shelf H_k holds residual values r satisfying $\varepsilon_{k-1} > r \geq \varepsilon_k$. Residual entries smaller than ε_N are omitted from H (since convergence does not require operating on them). Residual entries with values greater than ε_0 are simply placed in shelf H_0.

PPR on a grid of ε parameters.

During the iterative step of `ppr-grid`, then, rather than place a residual entry at the back of Q, we instead place the entry at the back of the appropriate shelf, H_k. Once all shelves $H_m(\mathbf{r})$ are cleared for $m \leq k$, then the residual has no entries larger than ε_k, and so we have arrived at an approximation vector satisfying convergence criterion (2) with accuracy ε_k. At this point, we perform a sweep procedure using the ε_k-solution. We then repeat the process until the next shelf is cleared, and a new ε_{k+1}-solution is produced.

PPR grid algorithm. The iterative step is as follows:

```
1. Determine the top-most non-empty shelf, $H_k$.
2. While $H$ contains an entry in shelf $k$ or above, do the following:
    2.1. Pop an entry on or above shelf $H_k$, say value $r$ in entry $\mathbf{r}_j$, and set $\mathbf{r}_j = 0$.
    2.2. Add $r$ to $\mathbf{x}_j$.
    2.3. Add $r\alpha\mathbf{P}^T\mathbf{e}_j$ to $\mathbf{r}$.
    2.4. For each entry of $\mathbf{r}$ that was updated, move that node to the correct shelf, $H_m$, where $\varepsilon_{m-1} > r \geq \varepsilon_m$. If an entry is placed on a shelf higher than $k$, record the new top-shelf.
3. Shelves 0 through $k$ are cleared, so the $\varepsilon_k$-solution is done; perform a sweep.
```

Once all shelves are empty, the approximation with strictest accuracy, ε, has been attained, and a final sweep procedure is performed.

Shelf computation. In each iteration of `ppr-grid` we must place multiple entries into their respective "shelves". Here we show that computing the correct shelf where a value r will be placed can be accomplished in constant time.

Let $\varepsilon_k = \varepsilon_0\theta^k$ for a fixed value of $\theta \in (0,1)$. We want a value r satisfying $\varepsilon_{k-1} > r \geq \varepsilon_k$ to be placed on shelf k. If

$r \geq \varepsilon_0$, then we place r into shelf 0. Otherwise, making the substitution $\varepsilon_k = \varepsilon_0 \theta^k$ and performing some algebra yields

$$k - 1 < \frac{\log(r/\varepsilon_0)}{\log(\theta)} \leq k,$$

so k can be computed by taking the ceiling of $\log(r/\varepsilon_0)/\log(\theta)$, which is a constant time operation. Note that this process requires that $0 < \varepsilon_k < 1$ holds for all k, that $\theta \in (0, 1)$, and that $r > 0$.

Top shelf. Each step of `ppr-grid` also requires determining the top non-empty shelf. This can be done in constant time by tracking what the top shelf is during each residual update. If k is the top shelf immediately prior to step (2.4), then k will still be the top shelf after the residual update is complete, unless one of the updates in step (2.4) moves an entry to a shelf $l < k$. By checking for this event during the update of each individual residual entry in step (2.4), we will have knowledge of the top non-empty shelf at the beginning of each step, with only constant work per step.

Once the current working shelf is emptied, then it is possible that the next non-empty shelf is many shelves down, i.e. shelves H_k and higher are emptied and the next non-empty shelf is H_{k+c} for some large number c. Then determining $k + c$ takes $O(c)$ operations. However, this operation is performed every time the algorithm switches from one value of ε_k to the next. If there are N values of ε_k, then the total work in all calls of this top-shelf computation is bounded by $O(N)$.

THEOREM 2. *Given a random walk transition matrix $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{D}^{-1}$, stochastic vector $\boldsymbol{v}$, and input parameters $\alpha, \theta \in (0, 1)$ and $\varepsilon_k = \varepsilon_0 \theta^k$ with $\varepsilon_N = \varepsilon$, our* `ppr-grid` *algorithm outputs the best-conductance set found from sweeps over ε_k-accurate degree-normalized solution vectors $\hat{\boldsymbol{x}}$ to $(\boldsymbol{I} - \alpha \boldsymbol{P})\boldsymbol{x} = (1-\alpha)\boldsymbol{v}$, for all values ε_k for $k = 0$ through N. The work in computing the diffusions is bounded by $O(\frac{1}{\varepsilon(1-\alpha)})$. This improves on the method of computing the N diffusions separately, which is bounded by $O\left(\frac{1}{\varepsilon(1-\alpha)(1-\theta)}(1-\theta^{N+1})\right)$. The two methods perform the same amount of sweep-cut work.*

Proof. Note that the amount of push-work required to produce a diffusion with smallest accuracy ε is exactly the same as the push-work performed in computing an ε solution via `ppr-path`; The only difference is in how we organize the residual and solution vectors. Hence, the push-work for `ppr-grid` is bounded by $O(\varepsilon^{-1}(1-\alpha)^{-1})$. Updating the shelf structure for `ppr-grid` requires only a constant number of operations in each iteration, and so the dominating operation in one step of `ppr-grid` is the residual push work. Thus, the push-work bound for `ppr-grid` is $O(\varepsilon^{-1}(1-\alpha)^{-1})$.

Push-work for N separate diffusions. As noted above, computing a diffusion with parameters ε_k and α requires push-work $O(\varepsilon_k^{-1}(1-\alpha)^{-1})$. Summing this over all values of ε_k gives $\sum_{k=0}^{N} \varepsilon_k^{-1}(1-\alpha)^{-1} = (1-\alpha)^{-1}\sum_{k=0}^{N}(1/\varepsilon_k)$. Substituting $\varepsilon_0\theta^k$ in place of ε_k, we see this sum is simply a scaled partial geometric series, $\sum_{k=0}^{N} \varepsilon_k^{-1} = \varepsilon_0^{-1}\theta^{-N}(1-\theta^{N+1})/(1-\theta)$. Simplifying gives

$$\sum_{k=0}^{N} \frac{1}{\varepsilon_k(1-\alpha)} = \frac{1}{\varepsilon(1-\alpha)(1-\theta)}\left(1-\theta^{N+1}\right),$$

proving the bound on the push-work. For our choices $\varepsilon_0 = 10^{-1}$, $\varepsilon = 10^{-6}/3$, and $\theta = 0.66$ (which correponds to using $N = 32$ diffusions), this quantity is roughly 2.9 times greater than computing only one diffusion, as our method does.

Sweep work. The number of operations required in computing the diffusion is bounded by $O(\varepsilon^{-1}(1-\alpha)^{-1})$, but this does not include the work done in sweeping over the various ε_k-approximation vectors. The sweep operation requires sorting the solution vector. As noted in the proof of work for `ppr-path`, the number of nonzeros in the solution vector is bounded by $O(\varepsilon^{-1}(1-\alpha)^{-1})$, and so the sorting work is $O(\varepsilon^{-1}(1-\alpha)^{-1}\log(\varepsilon^{-1}(1-\alpha)^{-1}))$. This implies that sorting is the dominant subroutine of the algorithm. In practice the bound on the number of nonzeros in the solution is loose, and the push operations comprise most of the labor.

Table 1: Datasets

Graph	$\lvert V\rvert$	$\lvert E\rvert$	d_{ave}
`itdk0304`	190,914	607,610	6.37
`dblp`	226,413	716,460	6.33
`com-youtube`	1,134,890	2,987,624	5.27
`fb-one`	1,138,557	4,404,989	3.9
`fbA`	3,097,165	23,667,394	15.3
`ljournal`	5,363,260	50,030,085	18.5
`hollywood`	1,069,126	56,306,653	105
`twitter`	41,652,230	1,202,513,046	57.7
`friendster`	65,608,366	1,806,067,135	55.1

5. EXPERIMENTAL RESULTS

We have presented two frameworks for computing a single personalized PageRank diffusion across multiple parameter settings. Here we analyze their performance on a set of real-world social and information networks with varying sizes and edge-densities. All datasets were altered to be symmetric and have 0s on their diagonals; this is done by making all directed edges undirected, and deleting any self-edges. In addition to versions of the Facebook dataset analyzed in Section 2, we test our algorithms on graphs including twitter-2010 from [19], friendster from [34], dblp-2010 and hollywood-2009 in [4, 3], idk0304 from [29], and ljournal-2008 in [7]. See Table 1 for a summary of their properties.

Our first method, `ppr-path`, is aimed at studying how PPR diffusions vary with the parameter ε. Toward this, Table 2 emphasizes the shear volume of distinct ε-approximations that `ppr-path` explores. We also want to highlight both the efficiency of our method over the naive approach for computing the solution paths, and the additional information that the solution paths provide compared to a single diffusion.

With this in mind, our experiment proceeds as follows. On each data set, we selected 100 distinct nodes uniformly at random, and ran three personalized PageRank algorithms from that node, with the settings $\alpha = 0.99$ and $\varepsilon = 10^{-5}$. Table 2 displays results for our solution paths algorithm ("path" in the table) compared with two other algorithms chosen to emphasize the runtime and the performance of `ppr-path`.

To show how `ppr-path` scales compared to the runtime of a single diffusion, and to emphasize that the solution paths can locate better conductance sets in some cases, we compare with a standard implementation for computing a single PPR diffusion ("single" in Table 2). Column 3 in the table gives the median runtime, taken over 100 trials, of the single diffusion. To compare, column 4 gives the median *ratio* of "path" time to "single" time. Although `ppr-path` is slower on the small

Data	num ε	Time			Conductance	
		single	path r.	mult r.	single	best r.
itdk0304	5332	0.02	20.19	4336	0.11	0.56
dblp	8134	0.02	24.18	4502	0.12	0.89
fb-one	3474	0.01	5.72	4042	0.56	0.92
fbA	862	$< .01$	1.94	3817	0.69	0.86
ljournal	2795	0.01	3.66	4450	0.39	0.48
hollywood	382	$< .01$	1.13	4043	0.48	0.90
twitter	158	$< .01$	1.51	3300	0.84	0.95
friendster	408	$< .01$	1.56	3866	0.88	0.96

Table 2: Runtime and conductance comparison of the solution paths (all accuracies from 10^{-1} to 10^{-5}) with (1) a single PPR diffusion with accuracy 10^{-5} (labelled "single") and (2) 10,000 PPR diffusions, accuracies k^{-1} for $k = 1$ to 10,000 (labelled "mult"). An "r." denotes that the quantity displayed is a ratio of the performance of the indicated method with the single diffusion's performance. Column two displays the median number of distinct accuracy parameters ε explored by our algorithm `ppr-path`. On each dataset we selected 100 distinct nodes uniformly at random and ran the algorithms with the settings $\alpha = 0.99$ and $\varepsilon = 10^{-5}$. Results reported are medians over these 100 trials, except the column "best r." which lists the smallest (best) ratio of conductance achieved by our `ppr-path` with conductance achieved by a single diffusion.

graphs, on the larger graphs we see the runtime is nearly the same as for a single PPR diffusion. At the same time, column 2 shows that "path" computes the results from hundreds or even thousands of diffusions, a significant gain in information over the single PPR diffusion. Finally, column 7 gives the best ratio of conductance found by "path" compared to that found by "single". This shows that the solution paths can improve conductance by 10% to even 50%.

To display the efficiency of our algorithm in computing these many diffusion settings, we again use the standard PPR implementation, but this time set to compute the diffusion for every accuracy setting k^{-1} for $k = 1$ to $10,000$. This algorithm is "mult" in Table 2, and is essentially a naive method for approximating the solution paths. Column 5 gives the ratio of "mult" time to "single" time, and shows that this naive approach to computing diffusions with multiple accuracies is prohibitively slow – it is thousands of times slower than our "path" method.

Lastly, we acknowledge here that both variations on the PPR diffusion are naive approaches to the problem at hand. However, currently there is no other algorithm for computing the PPR solution paths which we can use as a more fair baseline.

5.1 Runtime and Conductance: `ppr-grid`

We compare our second method `ppr-grid` with a method called `ppr-grow`, which uses the push framework described in Section 4.1. Both of these algorithms uses a variety of accuracy settings, and returns the set of best conductance found from performing a sweep-cut over the diffusion vector resulting from each accuracy setting. The algorithm `ppr-grow` has 32 pre-set accuracy parameters ε_k. In contrast with `ppr-grid`, which takes its accuracy parameters from a log-spaced grid $\varepsilon_k = \varepsilon_0\theta^k$, the parameters for `ppr-grow` are chosen as the inverses of values from the grid $10^j \cdot \begin{bmatrix} 2 & 3 & 4 & 5 & 10 & 15 \end{bmatrix}$ for $j = 0, 1, \cdots, 4$, along with two additional parameters, $10^{-6}/2$ and $10^{-6}/3$.

In addition to α, our method `ppr-grid` has the parameters ε_0 and ε, the laxest and strictest accuracies (respectively), and θ, which determines the fineness of the grid of accuracy parameters. We use the values $\varepsilon_0 = 10^{-1}$ and $\varepsilon = 10^{-6}/3$, and use values of θ corresponding to $N = 32$, 64, and 1256 different accuracy parameters.

We emphasize that this comparison with the `ppr-grow` method is not as naive as it might seem: out of the 32 calls that it makes, in practice the very last call (with the strictest value of ε) constitutes near 37% of the total runtime, so making only a single call saves little work, and at the expense of the information from the other 31 (smaller) approximations. Furthermore, the primary optimizations that would be made to the `ppr-grow` framework to improve on this are exactly the optimizations that we make with our `ppr-grid` algorithm.

Because the two algorithms compute the same PageRank diffusion, comparing their runtimes here allows us to study what proportion of the total work is made up of redundant push operations, and what proportion is comprised of the sweep cut procedures, which both algorithms perform anew for each diffusion. To study this, we highlight the results in Table 3 which displays the runtimes for `ppr-grow` and the *ratios* of the runtimes of `ppr-grow` with `ppr-grid` for computing the best-conductance set from the same number of different diffusions, $N = 32$. We also display `ppr-grid` results for the cases $N = 64$ and 1256 to show how the algorithm scales with the fineness of the grid.

To compare runtimes, we perform the following for each different dataset. For 100 distinct nodes selected uniformly at random, we ran both algorithms with the setting $\alpha = 0.99$. We display the best (25%) and worst (75%) quartile of performance of each algorithm and parameter setting. In almost all cases, we see that `ppr-grid` with $N = 32$ has a speedup of a factor 2 to 3. This is consistent with our theoretical comparison of the two runtimes in Theorem 2, which predicts a factor of 2.9 difference in the push-work that the two algorithms perform.

The conductances displayed in Table 4 are taken from the same trials as the runtime information in Table 3. As with the table of runtimes, for each dataset the table gives the 25% (best) and 75% (worst) percentiles of conductance scores produced by each algorithm on the 100 trials. We see nearly identical conductance scores for `ppr-grow` and `ppr-grid` with $N = 32$, which we expect because the two perform nearly identical work. It is interesting to note, however, that increasing the number of diffusions can result in significantly improved conductance scores in some cases, as with $N = 1256$ on the "fb-one" and "hollywood" datasets. This demonstrates concretely the potential effect of using a broad swath of parameter settings for ε. Moreover, it demonstrates that even a finely spaced mesh of ε values, as with `ppr-grow` and `ppr-grid` with $N = 64$, can miss informative diffusions.

6. RELATED WORK

There is a variety of work that considers the PageRank vector as a function of the teleportation parameter α [5, 20]. Much of this work seeks to understand the sensitivity of the problem with respect to α. For instance, we can compute

Data	ppr-grow		$N = 32$		$N = 64$		$N = 1256$	
	25	75	25	75	25	75	25	75
itdk0304	6.42	9.06	0.56	0.60	0.61	0.64	1.12	1.18
dblp	4.80	7.83	0.53	0.63	0.59	0.69	1.24	1.44
fb-one	1.46	1.96	0.33	0.38	0.44	0.51	3.65	4.33
fbA	0.55	0.74	0.46	0.54	0.62	0.70	5.93	6.71
ljournal	0.83	1.29	0.43	0.54	0.57	0.72	4.57	5.95
hollywood	0.33	1.07	0.36	0.52	0.46	0.66	3.33	5.15
twitter	0.17	0.44	0.41	0.45	0.55	0.62	4.41	5.47
friendster	0.31	0.44	0.39	0.45	0.52	0.60	4.09	4.58

Table 3: Runtime comparison of our ppr-grid with ppr-grow. For each dataset, we selected 100 distinct nodes uniformly at random and ran ppr-grow with 32 and ppr-grid with N different accuracy settings ε_k. Columns 2 and 3 display the %25 and %75 runtimes for ppr-grow (in seconds). The other column displays the median over the 100 trials of the *ratios* of the runtimes of the indicated algorithm/parameter setting with the runtime of ppr-grid on the same node. These results demonstrate empirically that our algorithm with $N = 32$ achieves the factor of 2 to 3 speed-up predicted by our theory in Section 4.4.

Data	grow	$N = 32$		$N = 64$		$N = 1256$	
		25	75	25	75	25	75
itdk0304	0.06	1.00	1.00	0.99	1.00	0.98	1.00
dblp	0.07	1.00	1.00	1.00	1.00	0.99	1.00
fb-one	0.37	0.86	0.95	0.79	0.91	0.73	0.85
fbA	0.56	0.96	1.00	0.94	1.00	0.92	1.00
ljournal	0.32	0.99	1.00	0.99	1.00	0.99	1.00
hollywood	0.25	0.97	1.00	0.97	1.00	0.95	1.00
twitter	0.81	1.00	1.00	1.00	1.00	1.00	1.00
friendster	0.85	1.00	1.00	0.99	1.00	0.99	1.00

Table 4: Conductance comparison of our ppr-grid with ppr-grow. Column 2 displays the median of the conductances found by ppr-grow in the same 100 trials presented in Table 3. The other columns display the 25% and 75% percentiles of the *ratio* of the conductances achieved by ppr-grow and ppr-grid for the same seed set. For example, on the dataset 'fb-one', ppr-grid with $N = 1256$ accuracy settings found conductances 15% better than those found by ppr-grow — and that comparison is on the quartile of trials where ppr-grid compares the *worst* to ppr-grow.

the derivative of the PageRank vector with respect to α. It is also used to extrapolate solutions to accelerate PageRank methods [6]. More recently, varying α was used to show a relationship between personalized-PageRank-like vectors and spectral clustering [23].

As we already mentioned, regularization paths are common in statistics [8, 14], and they help guide model selection questions. In terms of clustering and community detection, solution paths are extremely important for a new type of convex clustering objective function [22, 16]. Here, the solution path is closely related to the number of clusters in the model.

As recently established by Ghosh et al. [9], there are many related diffusion methods that all share Cheeger-like inequalities for specific definitions of conductance. They also established local algorithms to compute them by adapting the push procedure. We anticipate that our path-tracking algorithm could apply to any of these diffusions as well. For instance, our recent result on estimating the heat kernel diffusion in large graphs is based on the push step as well [17]; we anticipate mild difficulty in adapting our results to that diffusion.

7. CONCLUSIONS AND DISCUSSION

We proposed two algorithms that utilize the push step in new ways to generate refined insights on the behavior of diffusions in networks. The first is a method to rapidly estimate the degree-normalized PageRank solution path as a function of the tolerance ε. This method is slower than estimating the solution of a single diffusion in absolute run time, but still fast enough for use on large graphs. We designed that method, and the associated degree-normalized PageRank solution path plot, in order to reveal new insights about local regions in large networks. The second method is a fast approximation to the solution path on a grid of logarithmically-spaced ε values. It uses an interesting application of bucket sort to efficiently manage these diffusions. We demonstrate that both of these algorithms are fast and local on large networks.

There is a variety of future work we consider in the spirit of this research. First, we hope to tackle the solution path problem for the exact regularized PageRank problem [11]. This is more difficult as solutions depends on a precise KKT condition. On the other hand, they are always unique unlike the paths we consider here. Second, we plan to extend the solution path framework to other diffusions as discussed in the related work section. Finally, there is much room to optimize our software implementations. We designed the ppr-grid method for *dense* solutions. On many massive graphs, however, the solutions do not densify before the method terminates (although we know they do eventually). We plan to dynamically switch between efficient sparse and dense codes. This will allow us to take advantage of processor caches and should provide a meaningful improvement in speed.

Acknowledgements

We thank the following people for their careful reading of several early drafts: Huda Nassar, Bryan Rainey, and Varun Vasudevan. This work was supported by NSF CAREER Award CCF-1149756.

8. REFERENCES

[1] R. Andersen and F. Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In *Theory and Applications of Models of Computation*, pages 1–12. 2007.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *FOCS*, 2006.

[3] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: Model and applications. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 609–618, New York, NY, USA, 2008. ACM.

[4] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th WWW2011*, pages 587–596, March 2011.

[5] P. Boldi, M. Santini, and S. Vigna. PageRank: Functional dependencies. *ACM Trans. Inf. Syst.*, 27(4):1–23, 2009.

[6] C. Brezinski, M. Redivo-Zaglia, and S. Serra-Capizzano. Extrapolation methods for pagerank computations. *Comptes Rendus Mathematique*, 340(5):393 – 397, March 2005.

[7] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 219–228, New York, NY, USA, 2009. ACM.

[8] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Ann. Statist.*, 32(2):407–499, 04 2004.

[9] R. Ghosh, S.-h. Teng, K. Lerman, and X. Yan. The interplay between dynamics and networks: Centrality, communities, and cheeger inequality. In *KDD*, pages 1406–1415, 2014.

[10] D. F. Gleich. PageRank beyond the web. *arXiv*, cs.SI:1407.5107, 2014.

[11] D. F. Gleich and M. M. Mahoney. Algorithmic anti-differentiation: A case study with min-cuts, spectral, and flow. In *ICML*, pages 1018–1025, 2014.

[12] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *KDD*, pages 597–605, Aug. 2012.

[13] T. Gutierrez-Bunster, U. Stege, A. Thomo, and J. Taylor. How do biological networks differ from social networks? (an experimental study). In *ASONAM*, pages 744–751, 2014.

[14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[15] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: Structural role extraction and mining in large graphs. In *KD*, pages 1231–1239, 2012.

[16] T. Hocking, J.-P. Vert, A. Joulin, and F. R. Bach. Clusterpath: an algorithm for clustering using convex fusion penalties. In *ICML)*, pages 745–752, 2011.

[17] K. Kloster and D. F. Gleich. Heat kernel based community detection. In *KDD*, pages 1386–1395, 2014.

[18] I. M. Kloumann and J. M. Kleinberg. Community membership identification from small seed sets. In *KDD*, 2014.

[19] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

[20] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

[21] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, September 2009.

[22] F. Lindsten, H. Ohlsson, and L. Ljung. Just relax and come clustering! a convexification of k-means clustering. Technical report, Linköpings universitet, 2011.

[23] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *Journal of Machine Learning Research*, 13:2339–2365, August 2012.

[24] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[25] B. L. Mooney, L. R. Corrales, and A. E. Clark. MoleculaRnetworks: An integrated graph theoretic and data mining tool to explore solvent organization in molecular simulation. *J. Comput. Chem.*, 33(8):853–860, 2012.

[26] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3):036104, September 2006.

[27] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[28] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS*, 105(35):12763–12768, 2008.

[29] C. (The Cooperative Association for Internet Data Analyais). Network datasets. http://www.caida.org/tools/measurement/skitter/router_topology/, 2005. Accessed in 2005.

[30] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[31] K. Voevodski, S.-H. Teng, and Y. Xia. Spectral affinity in protein networks. *BMC Systems Biology*, 3(1):112, 2009.

[32] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. In *CIKM*, pages 2099–2108, 2013.

[33] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *EuroSys*, pages 205–218, 2009.

[34] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 745–754, Dec 2012.

[35] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NIPS*, 2003.

[36] X.-N. Zuo, R. Ehmke, M. Mennes, D. Imperati, F. X. Castellanos, O. Sporns, and M. P. Milham. Network centrality in the human functional connectome. *Cerebral Cortex*, 2011.