Euro. Jnl of Applied Mathematics (2016), *vol.* 27, *pp.* 812–845. © Cambridge University Press 2016
doi:10.1017/S0956792516000280

Seeded PageRank solution paths

D. F. GLEICH[1] and K. KLOSTER[2]

[1]*Department of Computer Science, Purdue University, West Lafayette IN, USA*
email: dgleich@purdue.edu
[2]*Department of Mathematics, Purdue University, West Lafayette IN, USA*
email: kkloste@purdue.edu

(*Received 30 November 2015; revised 26 May 2015; accepted 27 May 2016; first published online 1 July 2016*)

We study the behaviour of network diffusions based on the PageRank random walk from a set of seed nodes. These diffusions are known to reveal small, localized clusters (or communities), and also large macro-scale clusters by varying a parameter that has a dual-interpretation as an accuracy bound and as a regularization level. We propose a new method that quickly approximates the result of the diffusion for all values of this parameter. Our method efficiently generates an approximate *solution path* or *regularization path* associated with a PageRank diffusion, and it reveals cluster structures at multiple size-scales between small and large. We formally prove a runtime bound on this method that is independent of the size of the network, and we investigate multiple optimizations to our method that can be more practical in some settings. We demonstrate that these methods identify refined clustering structure on a number of real-world networks with up to 2 billion edges.

Key words: Random walks on graphs, graphs and linear algebra (matrices, eigenvalues, etc.), programming involving graphs or networks, social networks, complex networks.

1 Introduction

Networks describing complex technological and social systems display many types of structure. One of the most important types of structure is clustering because it reveals the modules of technological systems and communities within social systems. A tremendous number of methods and objectives have been proposed for this task (survey articles include refs. [28, 32]). The vast majority of these methods seek large regions of the graph that display evidence of local structure. For the case of modularity clustering, methods seek statistically anomalous regions; for the case of conductance clustering, methods seek dense regions that are weakly connected to the rest of the graph. All of the objective functions designed for these clustering approaches implicitly or explicitly navigate a trade-off between cluster size and the underlying clustering signal. For example, large sets tend to be more anomalous than small sets. Note that these trade-offs are essential to multi-objective optimization, and the choices in the majority of methods are natural. Nevertheless, directly optimizing the objective makes it difficult to study these structures as they vary in size from small to large because of these implicit or explicit biases. This

intermediate regime represents the meso-scale structure of the network, and includes such network characteristics as the core-periphery property found in many real-world networks.

In this manuscript, we seek to study structures in this meso-scale regime by analysing the behaviour of seeded graph diffusions. Seeded graph diffusions model the behaviour of a quantity of "dye" that is continuously injected at a small set of vertices called the *seeds* and distributed along the edges of the graph. These seeded diffusions can reveal multi-scale features of a graph through their dynamics. The class we study can be represented in terms of a column-stochastic distribution operator $\mathbf{P}$:

$$\mathbf{x} = \sum_{k=0}^{\infty} \gamma_k \mathbf{P}^k \mathbf{s},$$

where γ_k are a set of diffusion coefficients that reflect the behaviour of the dye k steps away from the seed, and $\mathbf{s}$ is a sparse, stochastic vector representing the seed nodes. More specifically, we study the PageRank diffusions

$$\mathbf{x} = \sum_{k=0}^{\infty} (1-\alpha)\alpha^k \mathbf{P}^k \mathbf{s}.$$

The PageRank diffusion is equivalent to the stationary distribution of a random walk that: (i) with probability α, follows an edge in the graph and (ii) with probability $(1-\alpha)$ jumps back to a seed vertex (see Section 3 more detail on this connection).

PageRank itself has been used for a broad range of applications including data mining, machine learning, biology, chemistry, and neuroscience; see our recent survey [11]. Among all the uses of PageRank, the *seeded variation* is frequently used to localize the PageRank vector within a subset of the network; this is also known as *personalized PageRank* due to its origins on the web, or *localized PageRank* because of its behaviour.(We will use these terms: seeded PageRank, personalized PageRank, and localized PageRank interchangeably and use the standard acronym PPR to refer to them.) Perhaps, the most important justification for this use is presented in [2], where the authors determined a relationship between seeded PageRank 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 PageRank-based clustering method, called the PUSH method, has been used for a number of important insights into communities in large social and information networks [18,22].

Our focus is a novel application of this PUSH method for meso-scale structural analysis of networks. The PUSH method, which we'll describe formally in Section 4, depends on an accuracy parameter ε. As we vary ε, the result of the PUSH method for approximating the PageRank diffusion reveals different structures of the network. 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 that result from PUSH have only a few non-zeros for large values of ε. (Aside: 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.) This is interesting because an accurate PageRank vector is mathematically non-zero everywhere in the graph. PUSH, with large values of ε, produces sparse approximations to the PageRank vector. This connection is formal, and the parameter ε has a dual interpretation as a sparsity regularization parameter [12] (reviewed in Section 4.2). The solution path or regularization path for a parameter is the set of trajectories that

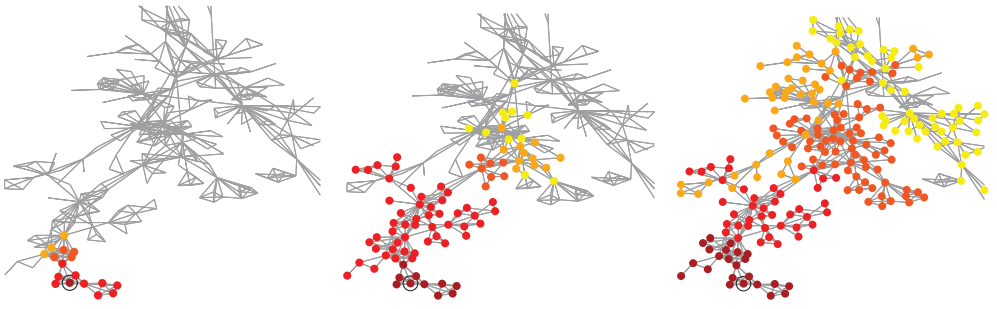

FIGURE 1. Nodes coloured by their degree-normalized PageRank values as ε varies: dark red is large, yellow is small. The hidden nodes are mathematically zero. As ε decreases, more nodes become non-zero. (a) $\varepsilon = 10^{-2}$, (b) $\varepsilon = 10^{-3}$ and (c) $\varepsilon = 10^{-4}$.

the components of the solution trace out as the parameter varies [9]. This interpretation motivates our study of solution paths for seeded PageRank diffusions.

1.1 Our contributions

We present new algorithms based on the PUSH *procedure* that allow us to approximate the solution path trajectories for seeded PageRank as a function of ε. We use our solution path approximation to explore the properties of graphs at many size-scales in Section 5, identifying features such as community strength and nested communities. In our technical description, we show that the solution path remains localized in the graph (Theorem 1). Experiments show that it runs on real-world networks with millions of nodes in less than a second (Section 7).

The PUSH method has become a frequently used graph mining primitive because of the sparsity of the vectors that result from when PUSH is used to approximate the seeded PageRank diffusion, along with the speed at which they can be computed. The method is typically used to identify sets of low-conductance (defined in Section 3) in a graph as part of a community or cluster analysis [10, 14, 15, 18, 22, 30]. In these cases, the insights provided by the solution paths are unlikely to be necessary. Rather, what is needed is a faster way to compute these diffusions for many values of ε. To address this difficulty, we propose a data structure called a *shelf* that we demonstrate can use 40 times as many values of ε in only seven times the runtime (Section 7.3).

In the spirit of reproducible research, we make all our codes publicly available: `https://github.com/kkloste/pagerank-paths`

2 Related work

As we already mentioned, regularization paths are common in statistics [9, 16], 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 [17, 23]. Here, the solution path is closely related to the number and size of clusters in the model.

One of the features of the solution path that we utilize to understand the behaviour

of the diffusion is the stability of the set of best conductance over time. In ref. [8], the authors use a closely related concept to study the persistence of communities as a different type of temporal relaxation parameter varies. Again, they use the stability of communities over regions of this parameter space to indicate high-quality clustering solutions.

In terms of PageRank, there is a variety of work that considers the PageRank vector as a function of the teleportation parameter α [5, 21]. Much of this work seeks to understand the sensitivity of the problem with respect to α. For instance, we can compute 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 [24]. Note that PageRank solution paths as α varies would be an equally interesting parameter regime to analyse. The parameter α functions akin to ε in that large values of α cause the diffusion to propagate further in the graph.

3 Technical preliminaries

We first fix our notation and review the Andersen–Chung–Lang procedure, which forms the basis for many of our contributions. 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, connected, and undirected. Let G have $n = |V|$ nodes and fix a labelling of the graph nodes using the numbers 1, 2, …, n. We refer to a node by its label. For each node j, we denote its degree by d_j.

The *adjacency matrix* of the graph G, which we denote by $\mathbf{A}$, is 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. The matrix $\mathbf{D}$ denotes the diagonal matrix with entry (i, i) equal to the degree of node i, d_i. Since G is connected, $\mathbf{D}$ is invertible, and we can define the *random walk transition matrix* $\mathbf{P} := \mathbf{A}\mathbf{D}^{-1}$.

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.

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 $\mathrm{vol}(S) = \sum_{j \in S} d_j$. Next, define the *boundary* of $S \subseteq V$ 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

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

Conductance can be thought of as measuring the extent to which a set is more connected to itself than the rest of the graph and is one of the most commonly used community detection objectives [28].

3.1 PageRank and Andersen–Chung–Lang method

The Andersen–Chung–Lang method uses PageRank vectors to identify a set of small conductance focussed around a small set of starting nodes [2]. We call such starting nodes *seed sets* and the resulting communities, *local* communities. We now briefly review this method starting with PageRank.

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

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}. \tag{3.1}$$

Note that when $\alpha \in (0, 1)$, the system in (3.1) can be solved via a Neumann series expansion, and so the solution $\mathbf{x}$ to this linear system is equivalent to the PageRank diffusion vector described in Section 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 *seeded* on the set S (or *personalized* on the set S).

Given PageRank diffusion scores $\mathbf{x}$, the Andersen–Chung–Lang procedure uses the values $\mathbf{x}_j/d_j$ to determine an order for a sweep-cut procedure (described below) that identifies a set of good conductance. Thus, we would like to bound the error in approximating the values $\mathbf{x}_j/d_j$. Specifically, for their theoretical guarantees regarding conductance to hold, we need our approximate solution $\hat{\mathbf{x}}$ to satisfy

$$0 \leqslant \mathbf{x}_j - \hat{\mathbf{x}}_j < \varepsilon d_j \qquad \text{or equivalently,} \qquad \mathbf{x} \geqslant \hat{\mathbf{x}}, \text{ and } \|\mathbf{D}^{-1}(\mathbf{x} - \hat{\mathbf{x}})\|_\infty < \varepsilon. \tag{3.2}$$

Once a PPR diffusion $\mathbf{x}$ is computed to this accuracy, a near-optimal conductance set located nearby the seed nodes is generated from the following a *sweep cut* procedure. Rank the nodes in descending order by their scaled diffusion scores $\mathbf{x}_j/d_j$, with large scores ranking the highest. Denote the set of nodes ranked 1 through m by $S(m)$. Iteratively compute the conductance of the sets $S(m)$ for $m = 2, 3, \ldots$, until $\mathbf{x}_m/d_m = 0$. Return the set $S(t)$ with the minimal conductance. This returned set is related to the optimal set of minimum conductance nearby the seed set through a localized Cheeger inequality [2]. The value of ε relates to the possible size of the set.

4 The PUSH PROCEDURE

The PUSH procedure is an iterative algorithm to compute a PageRank vector to satisfy the approximation (3.2). The distinguishing feature is that it can accomplish this goal with a sparse solution vector, which it can usually generate without ever looking at the entire graph or matrix. This procedure allows the Andersen–Chung–Lang procedure to run without ever looking at the entire graph. As we discussed in the introduction, this idea and method are at the heart of our contributions and so we present the method in some depth.

At each step, PUSH updates only a single coordinate of the approximate solution like a coordinate relaxation method. We will describe its behaviour in terms of a general linear system of equations. Let $\mathbf{Mx} = \mathbf{b}$ be a square linear system with 1s on the diagonal, i.e. $M_{i,i} = 1$ for all i. Consider an iterative approximation $\mathbf{x}^{(k)} \approx \mathbf{x}$ after k steps. The corresponding residual is $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Mx}^{(k)}$. Let j be a row index where we want to *relax*,

i.e. locally solve, the equation, and let r be the residual value there, $r = \mathbf{r}_j^{(k)}$. We update the solution by adding r to the corresponding entry of the solution vector, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + r\mathbf{e}_j$, in order to guarantee $\mathbf{r}_j^{(k+1)} = 0$. The residual can be efficiently updated in this case. Thus, the PUSH method involves the operations:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + r\mathbf{e}_j; \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - r\mathbf{M}\mathbf{e}_j. \end{aligned} \tag{4.1}$$

Note that the iteration requires updating just one entry of $\mathbf{x}^{(k)}$ and accessing only a single column of the matrix $\mathbf{M}$. It is this local update that enables PUSH to solve the seeded PageRank diffusion efficiently.

4.1 The Andersen–Chung–Lang PUSH procedure for PageRank

The full algorithm for the PUSH method applied to the PageRank linear system to compute a solution that satifies (3.2) for a seed set S is:

(1) Initialize $\mathbf{x} = 0, \mathbf{r} = (1-\alpha)\mathbf{e}_S$ using sparse data structures such as a hash-table.
(2) Add any coordinate i of $\mathbf{r}$, where $\mathbf{r}_i \geqslant \varepsilon d_i$ to a queue Q.
(3) While Q is not empty
(4) Let j be the coordinate at the front of the queue and pop this element.
(5) Set $\mathbf{x}_j = \mathbf{x}_j + \mathbf{r}_j$
(6) Set $\delta = \alpha\mathbf{r}_j/d_j$
(7) Set $\mathbf{r}_j = 0$
(8) For all neighbours u of node j
(9) Set $\mathbf{r}_u = \mathbf{r}_u + \delta$
(10) If $\mathbf{r}_u$ exceeds εd_u after this change, add u to Q.

The queue maintains a list of all coordinates (or nodes) where the residual is larger than εd_j. We choose coordinates to relax from this queue. Then, we execute the PUSH procedure to update the solution and residual. The residual update operates on only the nodes that neighbour the updated coordinate j. Once elements in the residual exceed the threshold, they are entered into the queue.

We have presented the PUSH method so far from a linear solver perspective. To instead view the method from a graph diffusion perspective, think of the solution vector as tracking where "dye" has concentrated in the graph and the residual as tracking where "dye" is still spreading. At each step of the method, we find a node with a sufficiently large amount of dye left (Step 4), concentrate it at that node (Step 5), then update the amount of dye that is left in the system as a result of concentrating this quantity of dye (Lines 6–10). The name PUSH comes from the pattern of concentrating dye and *pushing* newly unprocessed dye to the adjacent residual entries.

Note that the value of ε plays a critical role in this method as it determines the entries that enter the queue. When ε is large, only a small number of coordinates or nodes will ever enter the queue. This will result in a sparse solution. As $\varepsilon \to 0$, there

will be substantially more entries that enter the queue. In the original description of the Andersen–Chung–Lang algorithm, the authors proved that the PUSH method computes the diffusion with accuracy ε in work bounded by $O(\varepsilon^{-1}(1-\alpha)^{-1})$, and so the solution can have no more than $O(\varepsilon^{-1}(1-\alpha)^{-1}$ non-zeros. The convergence theory (and, hence, sparsity) of our novel method depends on constants which we have not yet introduced, and so for bounds on our proposed methods we direct the interested reader to Section 5.

4.2 Implicit regularization from PUSH

To understand the sparsity that results from the PUSH method, we introduce a slight variation on the standard PUSH procedure inspired in part by Andersen's interesting implementation of this algorithm. Rather than using the full update $\mathbf{x}_j + \mathbf{r}_j$ and pushing $\alpha\mathbf{r}_j/d_j$ to the adjacent residuals, we consider a method that takes a partial update. The form we assume is that we will leave $\varepsilon d_j \rho$ "dye" remaining at node j. For $\rho = 0$, this corresponds to the PUSH procedure described above. For $\rho = 1$, this update will remove node j from the queue, but PUSH as little mass as possible to the adjacent nodes such that the dye at node j will remain below εd_j. Andersen used $\rho = 0.5$, possibly as a means to increase the sparsity of the solution. The change is just at steps 5–7:

(5′) Set $\mathbf{x}_j = \mathbf{x}_j + (\mathbf{r}_j - \varepsilon d_j \rho)$

(6′) Set $\delta = \alpha(\mathbf{r}_j - \varepsilon d_j \rho)/d_j$

(7′) Set $\mathbf{r}_j = \varepsilon d_j \rho$

In previous work [12, Theorem 3], we showed that $\rho = 1$ produces a solution vector $\mathbf{x}$ that exactly solves a related 1-norm regularized optimization problem. The form of the problem that $\mathbf{x}$ solves is most cleanly stated as a quadratic optimization problem in $\mathbf{z}$, a degree-based rescaling of the solution variable $\mathbf{x}$:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\mathbf{z}^T\mathbf{Q}\mathbf{z} - \mathbf{z}^T\mathbf{g} + C\varepsilon\|\mathbf{D}\mathbf{z}\|_1 \\ &\text{subject to } \mathbf{z} \geqslant 0. \end{aligned} \tag{4.2}$$

The details of the optimization problem that are required to demonstrate the equivalence between $\mathbf{z}$ and $\mathbf{x}$ are tedious to state and uninformative to our purposes in this work. The important point is that ε can also be interpreted as a regularization parameter that governs the sparsity of the solution vector $\mathbf{x}$. Large values of ε increase the magnitude of the 1-norm regularizer and thus cause the solutions to be sparser. Moreover, the resulting solutions are unique as the above problem is strongly convex.

In this work, we seek algorithms to compute the solution paths or regularization paths that result from trying to use all values of ε to fully study the behaviour of the diffusion. In the next section, we explore some potential utilities of these paths before presenting our algorithms for computing them in Section 6.

5 Personalized PageRank paths

In this section, we aim to show the types of insights that the PageRank solution paths can provide. We should remark 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 study these patterns in the graph. Thus, much of the analysis here will be qualitative. In subsequent sections, we present our algorithm for computing these solution paths, as well as the quantitative advantages of the path methodology. We emphasize that these PageRank solution paths exist separately from our algorithms: Our method is simply one manner of computing the solution paths efficiently. Therefore, no understanding of our algorithm is required to be able to draw insights from the solution paths themselves.

The focus of this section is to illustrate the kinds of information that the solution paths can uncover near a seed node. With this in mind, the examples that we present use seeds that we selected as follows. First, we randomly sampled dozens of seeds from different datasets and determined what kinds of insights their solution paths offered. Then, we chose a subset of those seeds as case studies for the kinds of features we noticed in the solution paths. Those are the seeds used in the figures below.

5.1 Exact paths and fast path approximations

As discussed in Section 4, the solution paths for displaying the behaviour of PageRank as ε varies have a single, exact, unique solution. We can obtain the exact solution path for the seeded PageRank diffusion by solving the regularized optimization problem (4.2) itself for all values of ε. This could be accomplished by using ideas similar to those used to compute solution paths for the Lasso regularizer [9]. Our algorithms and subsequent analysis evaluate *approximate* solution paths that result from using our PUSH-based algorithm with $\rho = 0.9$ (Section 6.2). In this section, we compare our approximate paths to the exact paths. We find that, while the precise numbers change, the qualitative properties are no different.

Figure 2 shows the results of such a comparison on Newman's netscience dataset (379 nodes, 914 edges [26]). Each curve or line in the plot represents the value of a non-zero entry of an approximate PageRank vector $\mathbf{x}_\varepsilon$ as ε varies (horizontal axis). As ε approaches 0 (and $1/\varepsilon$ approaches ∞), each approximate PageRank entry approaches its exact value in a monotonic manner. Alternatively, we can think of each line as the diffusion value of a node as the diffusion process spreads across the graph. The plots in the figure are for a PageRank vector from the same seed, using $\alpha = 0.99$.

The left plot was computed by solving for the optimality conditions of (4.2); the right plot was computed using our PPR path approximation algorithm (described in detail in Section 6.2), with $\rho = 0.9$. The values of ε are automatically determined by the algorithm itself. The plots show that our approximate paths and the exact paths have essentially identical qualitative features. For example, they reveal the same bends and inflections in individual node trajectories, as well as large gaps in PageRank values. The maximum difference between the two paths never exceeds 1.1×10^{-4}. The lower row of plots displays the behaviour of the best conductance attained as ε varies. These conductance plots also show that the approximate paths (right plot) reveal nearly exactly the same information as the exact paths (left plot).

These results were essentially unchanged for a variety of other sample diffusions we considered, and so we decided that using $\rho = 0.9$ was an acceptable compromise between

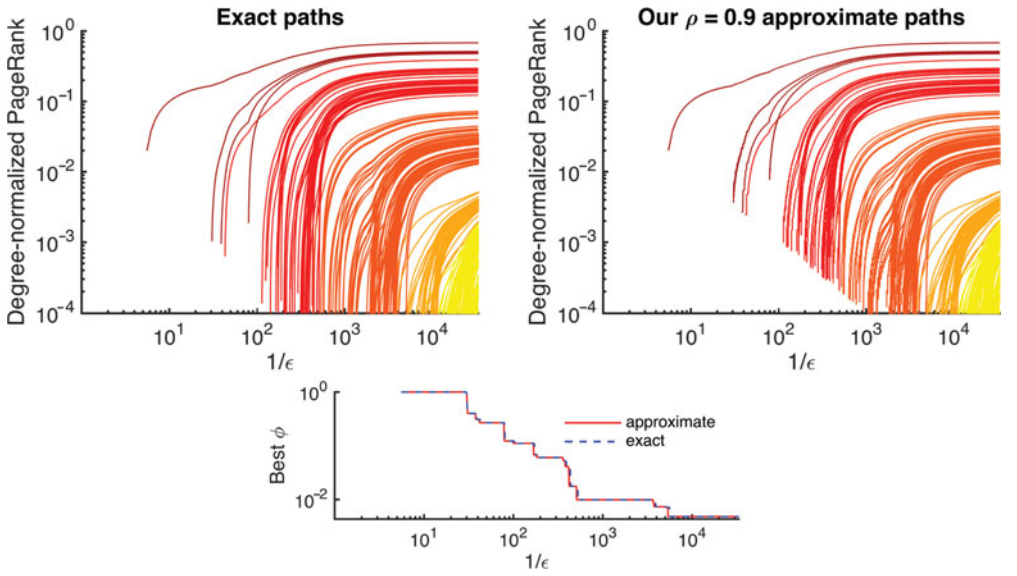

FIGURE 2. This figure compares the exact and approximate paths for a seeded PageRank vector from the same seed node, with $\alpha = 0.99$. (*Top, left*) The solution paths for a PageRank diffusion on Newman's netscience dataset from a single seed node computed by exactly solving the regularized problem. (*Top, right*) The approximate solution paths computed by our PUSH-based solution path algorithm with $\rho = 0.9$. Each line traces a value $\mathbf{x}_j$ as ε varies. The maximum infinity-norm distance between the two paths is 1.1×10^{-4}, showing that $\rho = 0.9$ provides a good quantitative approximation. Moreover, the two plots highlight identical qualitative features—for example, the large gaps between paths, and the strange bend in the paths near $\varepsilon = 10^{-3}$. The colouring of the lines is based on the values at the smallest value of ε. The values of ε used were generated by the approximate algorithm itself and we computed the exact solution at these same values for comparison. (*Bottom row*) The bottom plot shows that the approximate and exact paths yield sets of nearly identical conductance. More specifically, for each value of ε, the plot displays the best conductance obtained by a sweep over the ε-accurate solution vector produced by the exact solution paths (dashed blue curve) and the approximate curve (solid red curve). The only differences between these two "conductance curves" occur at sharp drops in conductance, where one algorithm's conductance does not drop at the exact same value of ε as the other algorithm's conductance.

speed and exactness. Thus, all path plots in this paper were created with $\rho = 0.9$, unless noted otherwise. (For analysis of the *differences* of the exact paths and ρ-paths, and in particular, the behaviour of the ρ-approximate paths as ρ varies, see Figure 7 below.)

5.2 The seeded PageRank solution path plot

We now wish to introduce a specific variation on the solution path plot that shows helpful contextual information (we refer the reader to Figure 3 for an example). In the course of computation, our solution path algorithm identifies a set of values of ε (a few hundred to many thousands, depending on the underlying graph and settings for ρ, α, and the minimum ε used), where it computes a PageRank approximation that satisfies the solution criteria (6.2). At each of these values of ε, we perform a sweep-cut procedure to identify the set of best conductance induced by the current solution. In the solution

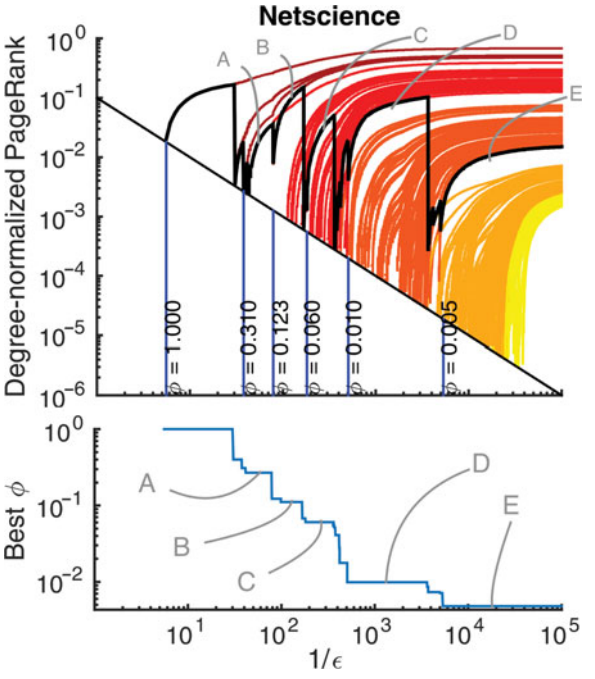

FIGURE 3. An example of the seeded PageRank solution path plot on Newman's netscience dataset. Each coloured line represents the value of a single node as the diffusion progresses from large ε to small ε. Because of our ρ-approximation to the true paths, the smallest value any node obtains is $(1-\rho)\varepsilon$ and we plot this as a thin, black, diagonal line. The thick black curve traces out the boundary of the set of best conductance found at each distinct value of ε as determined by a sweep-cut procedure. The vertical blue lines indicate values of ε, where set of minimum conductance reaches half of the conductance value of the previous blue line, and they are labelled with the conductance value. The colouring of the trajectory lines is based on the PageRank values at the smallest value of ε. Finally, whereas the thick black curve in the path plot indicates the *set of nodes* that attains the best conductance, the lower plot displays the *conductance value* that is attained by the set of nodes. The line denoting the set of best conductance is smooth and stable over certain intervals where the conductance value plateaus, indicating nested communities (manually annotated as *A*, *B*, *C*, *D*, and *E*). We discuss implications of the plot in Section 5.3.

path plot, we display the cut-point identified by this procedure as a thick black curve. All the nodes whose trajectories are above the dark black curve at a particular value of ε are contained in the set of best conductance at that value of ε. This curve allows us to follow the trajectory of the minimum-conductance set as we vary ε. We call this the "best-set curve".

Another property of our algorithm is that the smallest possible non-zero diffusion value in the solution is $(1-\rho)\varepsilon$. Thus, we plot this as a thin, diagonal, black line that acts as a pseudo-origin for all of the node trajectories. The vertical blue lines in the bottom left of the plot mark values of ε, where we detect a significant new set of best conductance. That is, each vertical blue line indicates that the best conductance attained has decreased by a factor of 2 since the previous vertical blue line.

Finally, we also include a plot (bottom) that shows how conductance varies with ε. More specifically, for each value of ε, the best conductance attained during a sweep over the PageRank vector $\mathbf{x}_\varepsilon$ is displayed. We call this plot the "conductance curve". Regions where the conductance curve stays flat indicate a set of best conductance that does not vary even as ε is varied. These regions correspond to intervals where the set of best conductance changes very little, or not at all. This can also be observed in the path plot as intervals where the best-set curve remains smooth and closely follows a single node's trajectory, rather than oscillating or dipping sharply. We interpret such good-conductance sets to be strong or "stable" communities, because no nodes enter or exit this set across a wide interval of ε values. Alternatively, such a "stable" set of best conductance can be understood as a set of nodes that does not change across an entire interval of the diffusion process.

5.3 Nested communities in netscience and Facebook

We now discuss some of the insights that arise from the solution path plot. In Figure 3, we show the seeded PageRank solution path plot for netscience with the settings $\alpha = 0.99$ and $\rho = 0.9$, and a minimum accuracy of 10^{-5}. With these settings, the plot displays information from 268,595 values of ε computed via our algorithm. (We remark that this number is quite large compared to other datasets. This occurs because netscience is such a small graph; on larger datasets the number of distinct ε values explored is in the hundreds to tens of thousands.) 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 good conductance. This behaviour 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. In particular, the path plots reveal multiple communities (good conductance sets) nested within one another through the gaps between the trajectories. Additionally, such nested communities can also be observed from the conductance curve (Figure 3, lower right). Intervals where the conductance curve is flat (marked as A, B, C, D, and E in the figure) indicate that the set of best conductance is identical across an entire interval of different ε values. At the same time, the thick curve indicating the best set smoothly follows a single node's trajectory over these intervals of ε, indicating that the set of nodes does not change, suggesting a good or "stable" community. Multiple intervals where the conductance curve is flat indicate multiple instances of such stable best-conductance sets nested within one another.

On a crawl of a Facebook network from 2009 where edges between nodes correspond to observed interactions [31] (see Table 1, `fb-one`, 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.) Pictured in Figure 4, 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. Furthermore, this example also shows evidence of multiple nested communities. These are illustrated with the manual

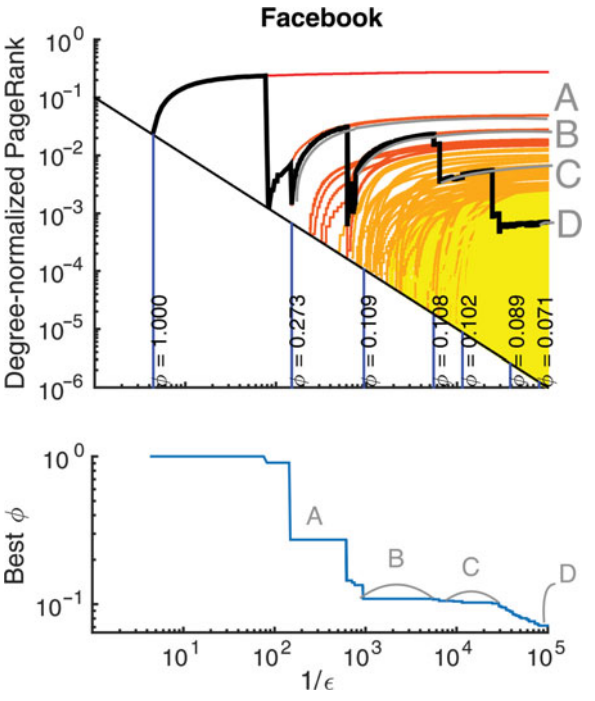

FIGURE 4. The seeded PageRank solution path for a crawl of observed Facebook network activity for 1 year (fb-one from Table 1) shows large, good cuts do not need to have large drops in the PageRank values. Nodes enter the solution and then quickly break into the best conductance set, showing that the frontier of the diffusion should be an interesting set in this graph. Furthermore, this path plot shows evidence of multiple nested communities (A through D), which were manually annotated. The set A is only a few nodes, but has a conductance score of 0.27; set B grows and improves this to a conductance of 0.1. The set C has nearly the same conductance as B, yet grows into a much larger set of nodes; and finally set D includes many more nodes to achieve a conductance of 0.07, which is an unusually small conductance value in a large social network.

annotations A, B, C, and the small curve D, which all indicate intervals of ε values, where the conductance curve is flat and the best-set curve is smooth, representing minimum-conductance sets that do not change as ε changes. These solution paths were computed using the settings $\alpha = 0.99$, $\rho = 0.9$, and a minimum accuracy of 10^{-5}.

5.4 Core and periphery structure in the US Senate

The authors in [18] analysed voting patterns across the first 110 US-Senates by comparing senators in particular terms (original data available here [27]). We form a graph from this US Senate data in which each senator is represented by a single node. For each term of the senate, we connect senators in that session with edges to their three nearest neighbours measured by voting similarities. This graph has a substantial temporal structure as a senator from 100 years ago cannot have any direct links to a senator serving 10 years ago. We show how our solution paths display markedly different characteristics when seeded on a node near the core of the network compared with a node near the periphery. This

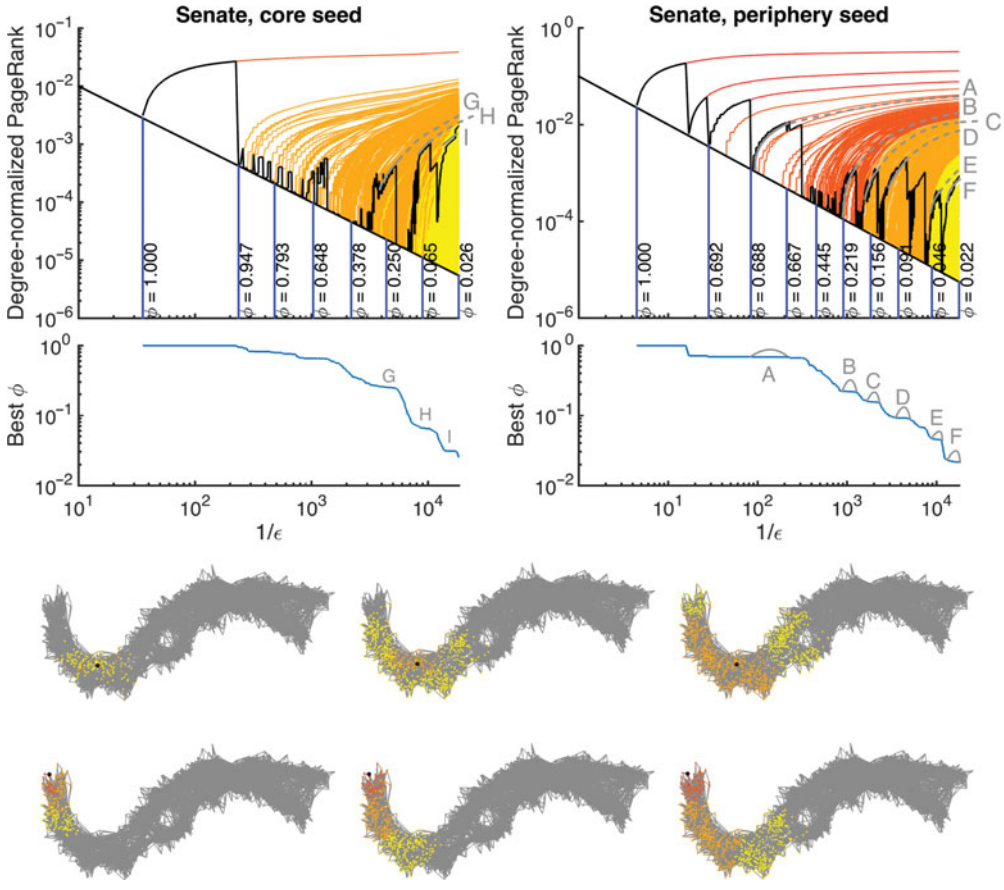

FIGURE 5. *(Top.)* The solution paths on the US-Senate graph for a senator in the core (who served multiple terms and is centrally located in a graph layout) and for a senator in the periphery (who served a single term and is located on the boundary of the graph layout). *(Bottom.)* The diffusions for each of these senators are shown as heat-plots on the graph layout. Red indicates nodes with the largest values and yellow the smallest. The seed nodes are circled in black in these layouts. The solution paths for a peripheral node indicate multiple nested communities, visible in the images of the diffusion on the whole graph and marked A through F. These sets are strongly correlated with successive terms of the Senate. In contrast, the core node diffusion appears to have only indicates one good cut. For the core node, we can see the diffusion essentially spreads across multiple dense regions simultaneously, without settling in one easily separated region until ε is small enough that the diffusion has spread to the entire left side of the graph. The sets F and I are also almost the same. (a) Core, $\varepsilon = 3\times\ 10^{-4}$, (b) Core, $\varepsilon = 10^{-4}$, (c) Core, $\varepsilon = 3\times\ 10^{-5}$, (d) Periphery, $\varepsilon = 3\times\ 10^{-4}$, (e) Periphery, $\varepsilon = 10^{-4}$ and (f) Periphery, $\varepsilon = 3\times\ 10^{-5}$.

example is especially interesting because the two different diffusions eventually lead to closely related cuts, as ε approaches 0.

Figure 5 displays solution paths seeded on a senator on the periphery of the network (top right) and a senator connected to the core of the network (top left). Here are some qualitative insights from the solution path plots. The peripheral seed is a senator who served a single term in the fifth Congress; the diffusion spreads across the graph slowly

because the seed is poorly connected to the network outside the seed senator's own senate term. As the diffusion spreads outside the seed's particular term, the paths identify multiple nested communities (manually annotated as sets A through F) that essentially reflect previous and successive terms of the Senate. We add the annotated sets based on intervals of ε, where the conductance curve plateaus and the best-set curve appears to be smooth.

In contrast, the core node is a senator who served eight terms between the 21st and 34th Congresses. The core node's paths skip over such smaller scale community structures (i.e. individual senate terms) as the diffusion spreads to each of those terms nearly simultaneously. Instead, the paths of the core node identify a few different cuts that appear to be nearly the same set of nodes (marked G, H, and I), with one particularly good cut (set I): The cut roughly separating all of the seed's terms from the remainder of the network. These solution paths were computed using the settings $\alpha = 0.99$, $\rho = 0.9$, and a minimum accuracy of 3×10^{-5}.

This example demonstrates the paths' potential ability to shed light on a seed's relationship to the network's core and periphery, as well as the seed's relationship to multiple communities.

5.5 Cluster boundaries in handwritten digit graphs

Finally, we use the solution paths to study the behaviour of a diffusion for a semi-supervised learning task. The USPS handwritten digits dataset consists of roughly 10,000 images of the digits 0 through 9 in human hand-writing [34]. Each digit appears in roughly 1,000 of the images, and each image is labelled accordingly. From this data, we construct a three-nearest-neighbours graph, and carry out our analysis as follows. Pick one digit, and select four seed nodes uniformly at random from the set of nodes labelled with this digit. Then, compute the PageRank solution paths from these seeds. Figure 6 shows the path plots with labels (right) and without (left). In the labelled plot, the correct labels are red and the incorrect labels are green. These solution paths used the settings $\alpha = 0.99$, $\rho = 0.9$, and a minimum accuracy of 10^{-4}.

We can use the best conductance set determined by the PPR vector to capture a number of other nodes sharing the seeds' label. However, this straight-forward usage of a PageRank vector results in a number of false positives. Figure 6 (right) shows that a number of nodes with incorrect labels are included in the set of best conductance (curves coloured green do not share the seed's label).

Looking at the solution-paths for this PageRank vector (Figure 6, left), we can see that a number of these false positives can be identified as the erratic lighter orange paths cutting across the red paths. Furthermore, the solution paths display earlier sets of best conductance (manually annotated as A) that would cut out almost all false positives. This demonstrates that the solution paths can be used to identify "stable" sets of best conductance that are likely to yield higher precision labelling results, even though they potentially have worse conductance than other sets found. Consequently, these results hint that a smaller, but more precise, set lurks inside of the set of best conductance. This information would be valuable when determining additional labels or trying to study new data that is not as well characterized as the USPS digits dataset.

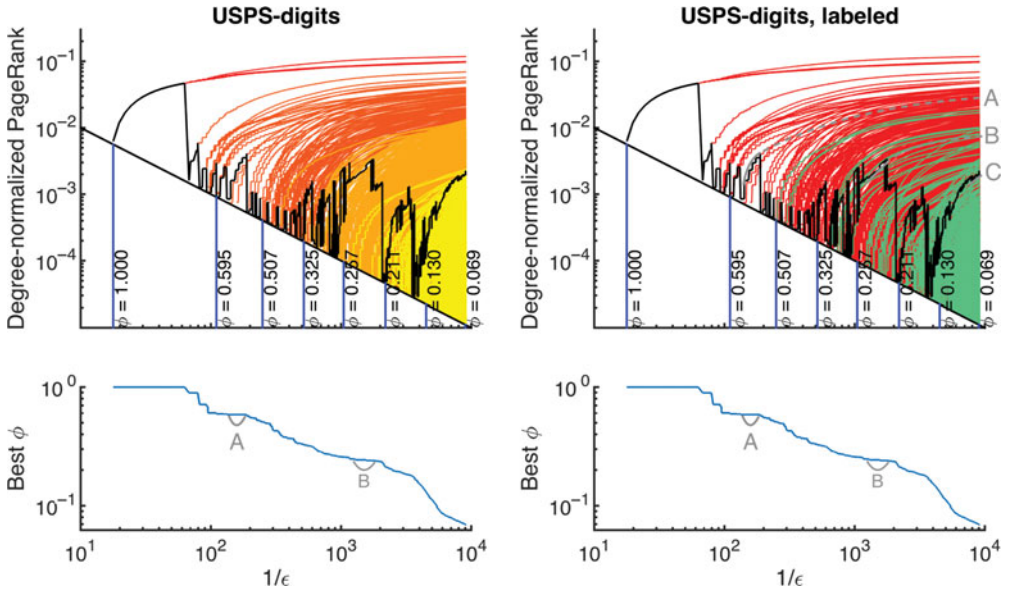

FIGURE 6. Seeded PageRank solution path plots for diffusions in the USPS digit dataset. The seeds are chosen to be images of handwritten digits with the same label. *(At left.)* The solution paths reveal a number of anomalous node trajectories near the set of best conductance. The black line indicating the set of best conductance oscillates up and down in many places, showing that the set of best conductance is erratic and frequently changes over those intervals. But two regions show a particularly flat conductance curve and smooth best-set curve (annotated as A and B) indicating a set of best conductance that is stable.*(At right.)* Here, we have coloured the solution path lines based on the true-class label. Red shows a correct label and green shows an incorrect label. We can see that the set achieving the best possible conductance (set C) does not have particularly good precision. Instead, stabler sets of good conductance, like A and B, obtain better precision.

This dataset is constructed using a parameter that must be chosen (i.e. k, in the k-nearest-neighbours graph), which raises the question of whether or not the behaviour of the solution paths depends on the parameter k. A recent work of ours explores this question of how different graph constructions affect the performance of graph algorithms; for example, how the choice of k in a k-NN graph affects the precision of a classification algorithm [13]. The conclusion is that the value of k that should be used is the value that is smallest while still preserving the signal of the underlying structure; we chose the smallest k such that the graph is still connected, $k = 3$.

5.6 Discussion

Overall, these seeded PageRank solution path plots reveal information about the clusters and sets near the seeds. Some of the features we've seen include nested community structure, core-periphery structure, and the stability of a community across an interval of a diffusion. They all provide refined information about the boundary of a community containing the seed, and suggest nodes with seemingly anomalous connections to the seed. For instance, some nodes enter the diffusion early but have only a slow-growing value indicating a weak connection to the seed; other nodes are delayed in entering the

diffusion but quickly grow in magnitude and end up being significant members of the cluster. Each of these features offers refined insights over the standard single-shot diffusion computation.

6 Algorithms

Here, we present two novel algorithms for analysing a PPR diffusion across a variety of accuracy parameter settings by computing the diffusion only a single time. Our first algorithm (Section 6.2) computes the best-conductance set from the ρ-approximate solution paths described in Section 4.2. Instead of obtaining the best-conductance set from a single PPR diffusion, this effectively finds the best-conductance set that can be obtained from *any* PPR diffusion that has accuracy in the interval $[\varepsilon_{\min}, \varepsilon_{\max}]$, where $\varepsilon_{\min}$ and $\varepsilon_{\max}$ are inputs. We prove the total runtime is bounded by $O(\varepsilon_{\min}^{-2}(1-\alpha)^{-2}(1-\rho)^{-2})$, though we believe improvements can be made to this bound. 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 (Section 6.3) in finding sets of low conductance, 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 $\varepsilon_0 > \varepsilon_1 > \cdots > \varepsilon_N$, 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 $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$; 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.

We begin by describing a modification to the PageRank linear system that will simplify our notation and the exposition of our algorithm.

6.1 A modified PageRank linear system for the PUSH PROCEDURE

Recall that the goal is to solve the PageRank linear system (3.1) to the accuracy condition (3.2) and then sort by the elements $\mathbf{x}_j/d_j$. If we multiply Equation (3.1) by $\mathbf{D}^{-1}$, then after some manipulation we obtain

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

Note this transformation relies on $\mathbf{A}$ being symmetric so that $\mathbf{P}^T = (\mathbf{A}\mathbf{D}^{-1})^T = \mathbf{D}^{-1}\mathbf{A} = \mathbf{D}^{-1}\mathbf{P}\mathbf{D}$. 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

$$(\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{y} = \mathbf{b}, \tag{6.1}$$

and we set $\mathbf{x}^{(k)} = (1-\alpha)\mathbf{D}\mathbf{y}^{(k)}$.

This connection between $\mathbf{x}$ and $\mathbf{y}$ enables us to establish a convergence criterion for our algorithms that will guarantee we obtain an approximation with the kind of accuracy typically desired for methods related to the PUSH operation, e.g. (3.2). More concretely, to guarantee $\|\mathbf{D}^{-1}(\mathbf{x}-\hat{\mathbf{x}})\|_\infty < \varepsilon$, it suffices to guarantee $\|\mathbf{y}-\hat{\mathbf{y}}\|_\infty < \frac{\varepsilon}{1-\alpha}$, so it suffices for our purposes to bound the error of the system (6.1).

The accuracy requirement has two components: non-negativity, and error. We relate the solution to its residual as the first step towards proving both of these. Left-multiplying the residual vector for (6.1) 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, if the right-hand side consists of all non-negative entries, then it is guaranteed that $\mathbf{y}-\mathbf{y}^{(k)} \geqslant 0$ holds. Recall from Section 4.1 that the residual update involved in the PUSH procedure consists of adding non-negative components to the residual, and so the residual *must* be non-negative. Then, since $(1-\alpha)\mathbf{y}^{(k)} = \mathbf{D}^{-1}\mathbf{x}^{(k)}$, this implies $\mathbf{x} \geqslant \mathbf{x}^{(k)}$, proving one component of the accuracy criteria (3.2) is satisfied.

Next, we bound the error in $\mathbf{y}$ in terms of its residual, and then control the residual's norm. Using the triangle inequality and sub-multiplicativity of the infinity norm allows us to bound $\|\mathbf{y}-\mathbf{y}^{(k)}\|_\infty$, which implies (3.2), with the following

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

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

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

So to guarantee $\mathbf{x}$ satisfies the desired accuracy, it is enough to guarantee that

$$\left\|\mathbf{r}^{(k)}\right\|_\infty < \varepsilon \tag{6.2}$$

holds, where $\mathbf{r}^{(k)} = \mathbf{b} - (\mathbf{I}-\alpha\mathbf{P}^T)\mathbf{y}^{(k)}$ and $\mathbf{x}^{(k)} = (1-\alpha)\mathbf{D}\mathbf{y}^{(k)}$. Thus, for our algorithms to converge to the desired accuracy, it suffices to iterate until the residual norm satisfies the bound (6.2). With this terminating condition established, we can now describe our algorithm for computing the solution paths of $\mathbf{x}_\varepsilon$ as ε varies.

6.2 PageRank solution paths

Recall that our goal is computing the solution paths of seeded PageRank with respect to the parameter ε. That is, we want an approximation $\mathbf{x}_\varepsilon$ of PageRank for all ε values inside some region. 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 $\varepsilon_{\min}$ and $\varepsilon_{\max}$. Then, for each value $\varepsilon_{\text{cur}} \in [\varepsilon_{\min}, \varepsilon_{\max}]$ (ε_{cur} denotes "the value of ε currently being considered"), we

want an approximation $\hat{\mathbf{y}}$ of the solution to $(\mathbf{I} - \alpha\mathbf{P}^T)\mathbf{y} = \mathbf{b}$ that satisfies $\|\mathbf{y} - \hat{\mathbf{y}}\|_\infty < \frac{\varepsilon_{\text{cur}}}{1-\alpha}$. (Or rather, we want a computable approximation to this information.) As discussed in Section 4.2, we also use the approximation parameter $\rho \in [0, 1)$ in the PUSH step.

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 $\mathbf{r}_j < \varepsilon_{\min}$. We store the entries of $Q(\mathbf{r})$ using a max-heap so that we can quickly determine $\|\mathbf{r}\|_\infty$ at every step. For readers less familiar with data structures and sorting operations—such as queues, heaps, and bubblesort—we provide brief descriptions in the Appendix.

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 $\|\mathbf{r}\|_\infty < \varepsilon_{\min}$ guarantees convergence to the desired accuracy. Next, we present the iteration in full detail.

6.2.1 *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, $\varepsilon_{\min}$.

(1) Pop the max of $Q(\mathbf{r})$, say entry j with value r, then set $\mathbf{r}_j = \rho\varepsilon_{\text{cur}}$ and re-heap $Q(\mathbf{r})$.
(2) Add $r - \rho\varepsilon_{\text{cur}}$ to $\mathbf{y}_j$.
(3) Bubblesort entry $\mathbf{y}_j$ in $L(\mathbf{y})$.
(4) If $L(\mathbf{y})$ changes, perform a local sweep update.
(5) Add $(r - \rho\varepsilon_{\text{cur}})\alpha\mathbf{P}^T\mathbf{e}_j$ to $\mathbf{r}$.
(6) For each entry i of $\mathbf{r}$ that was updated, if it does not satisfy $r_i < \varepsilon_{\min}$, then insert (or update) that entry in $Q(\mathbf{r})$ and re-heap.
(7) 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 $r_j < \varepsilon_{\min}$, and so our diffusion score approximations satisfy the accuracy requirement (3.2).

6.2.2 *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 non-zero 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 $1, 2, \ldots, m$ 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 $[1, \ldots, j - \Delta_k - 1]$, because the change in rankings happened entirely in the interval $[j - \Delta_k, \ldots, j]$. 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 $t = 0, \ldots, \Delta_k$) 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 neighbourhood of node $L(j)$ a single time. This provides a great savings in work compared to simply re-performing 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 $\alpha \in (0,1)$, $\rho \in [0,1)$, and $\varepsilon_{\max} > \varepsilon_{\min} > 0$, our* `ppr-path` *algorithm outputs the best-conductance set found from sweeps over ε_{cur}-accurate degree-normalized, ρ-approximate solution vectors $\hat{\boldsymbol{x}}$ to $(\boldsymbol{I} - \alpha\boldsymbol{P})\boldsymbol{x} = (1-\alpha)\boldsymbol{v}$, for all values $\varepsilon_{cur} \in [\varepsilon_{\min}, \varepsilon_{\max}]$. The total work required is bounded by $O\left(\frac{1}{\varepsilon_{\min}^2(1-\alpha)^2(1-\rho)^2}\right)$.*

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

PUSH WORK. Here, we count the operations performed on just the residual $\mathbf{r}^{(t)}$ and solution vector $\mathbf{y}^{(t)}$. The work required to maintain the heap Q and sorted array L is accounted for below.

In step t, the PUSH operation acts on a single entry in the residual that satisfies $r_t \geqslant \varepsilon_{\min}$. The step consists of a constant number of operations to update the residual and solution vectors (namely, updating a single entry in each). Let $j = j(t)$ denote the index of the residual that is operated on in step t. The actual amount that is removed from the residual node is $(r_t - \rho\varepsilon_{\min})$; then, we add $(r_t - \rho\varepsilon_{\min})$ to entry j in the solution, and $(r_t - \rho\varepsilon_{\min})\alpha/d_j$

to $\mathbf{r}_i^{(t)}$ for each neighbour i of node j. Since node j has d_j such neighbours, the total work in one step is bounded by $O\left(d_j\right)$. Suppose T total steps of the PUSH operation are performed in order to obtain an accuracy of $\varepsilon_{\min}$. Then, the amount of work required is bounded by $O(\sum_{t=0}^{T} d_{j(t)})$.

Next, we bound this expression for the work done in these "PUSH" steps, $O(\sum_{t=0}^{T} d_{j(t)})$. First, as noted above each step, we add a small amount to a particular entry of the solution vector $\mathbf{y}$: in step t, we add $(r_t - \rho\varepsilon_{\min})$ to entry j in the solution, $\mathbf{y}_j^{(t)}$. This means that at any given step, k, the solution can be expressed $\mathbf{y}^{(k)} = \sum_{t=0}^{k} \mathbf{e}_{j(t)}(r_t - \rho\varepsilon_{\min})$. But $\mathbf{x}$ and $\mathbf{y}$ are always related by $\mathbf{y}^{(k)} = (1/(1-\alpha))\mathbf{D}^{-1}\mathbf{x}^{(k)}$. Thus, using the expression for $\mathbf{y}$ above, we can write that

$$\mathbf{x}^{(k)} = (1-\alpha)\mathbf{D}\left(\sum_{t=0}^{k} \mathbf{e}_{j(t)}(r_t - \rho\varepsilon_{\min})\right) \tag{6.3}$$

$$= (1-\alpha)\sum_{t=0}^{k} \mathbf{e}_{j(t)} d_{j(t)}(r_t - \rho\varepsilon_{\min}). \tag{6.4}$$

Summing both sides at any step k yields $\mathbf{e}^T\mathbf{x}^{(k)} = (1-\alpha)\sum_{t=0}^{k} d_{j(t)}(r_t - \rho\varepsilon_{\min})$. We claim that the sum $\mathbf{e}^T\mathbf{x}^{(k)} \leqslant 1$ for all steps k, and in particular at the last step T. Assuming this for the moment, we get from Equation (6.4) that $(1-\alpha)\sum_{t=0}^{T}(r_t - \rho\varepsilon_{\min})\cdot d_{j(t)} = \mathbf{e}^T\mathbf{x}^{(T)} \leqslant 1$. Since each step of `ppr-path` operates on a residual value satisfying $r_t \geqslant \varepsilon_{\min}$, we know that $(r_t - \rho\varepsilon_{\min}) \geqslant \varepsilon_{\min}(1-\rho)$, and so

$$1 \geqslant (1-\alpha)\sum_{t=0}^{T}(r_t - \rho\varepsilon_{\min})\cdot d_{j(t)} \tag{6.5}$$

$$\geqslant (1-\alpha)\sum_{t=0}^{T}\varepsilon_{\min}(1-\rho)\cdot d_{j(t)}. \tag{6.6}$$

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

Lastly, we justify the claim $\mathbf{e}^T\mathbf{x}^{(k)} \leqslant 1$. Left-multiplying the equations in (6.1) by $(\mathbf{De})^T$ and using stochasticity of $\mathbf{v}$ gives

$$\begin{aligned}
\mathbf{e}^T(\mathbf{I} - \alpha\mathbf{P})\mathbf{D}\mathbf{y}^{(k)} &= \mathbf{e}^T\mathbf{D}\mathbf{b} - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)};\\
(1-\alpha)\mathbf{e}^T\tfrac{1}{(1-\alpha)}\mathbf{x}^{(k)} &= \mathbf{e}^T\mathbf{v} - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)};\\
\mathbf{e}^T\mathbf{x}^{(k)} &= 1 - \mathbf{e}^T\mathbf{D}\mathbf{r}^{(k)}.
\end{aligned} \tag{6.7}$$

All entries of the residual and iterative solution vector are non-negative at all times. The sum $\mathbf{e}^T\mathbf{x}^{(k)}$ cannot exceed 1 for any step k, then, because that would imply that the residual summed to a negative number, contradicting non-negativity of the residual vector. Hence, $\mathbf{e}^T\mathbf{x}^{(k)} \leqslant 1$ for all steps k.

Sorting and sweeping work. The proof of our work bound involves tedious details of data structures, and is less informative to our current purposes. We present the full proof of our work bound in the Appendix. □

6.3 Fast multi-parameter PPR

Here, we present a fast framework for computing ε-approximations of a PUSH-based 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. This algorithmic framework does not admit the parameter ρ as easily, because of implementation details surrounding the data structures used to handle sorting and updating the residual.

The framework is compatible with every set of parameter choices for ε 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 $\varepsilon_{k-1} > r \geqslant \varepsilon_k$. 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 grid formed from Chebyshev-like nodes, 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 PUSH method discussed in Section 4 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 call a shelf.

6.3.1 *The 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 $\varepsilon_{k-1} > r \geqslant \varepsilon_k$. 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.

6.3.2 *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 $m \leqslant k$, then the residual has no entries larger than ε_k, and so we have arrived at an approximation vector satisfying convergence criterion (3.2) 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:
(3) Pop an entry on or above shelf H_k, say value r in entry $\mathbf{r}_j$, and set $\mathbf{r}_j = 0$.
(4) Add r to $\mathbf{x}_j$.
(5) Add $r\alpha\mathbf{P}^T\mathbf{e}_j$ to $\mathbf{r}$.
(6) For each entry of $\mathbf{r}$ that was updated, move that node to the correct shelf, H_m, where $\varepsilon_{m-1} > r \geqslant \varepsilon_m$. If an entry is placed on a shelf higher than k, record the new top-shelf.
(7) Shelves 0 through k are cleared, so the ε_k-solution is done; perform a sweep.

Once all shelves are empty, the approximation with strictest accuracy, ε_N, has been attained, and a final sweep procedure is performed. For more details on how the shelf structure works, for example, how to determine on which shelf an entry should be placed, see the Appendix.

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 $\varepsilon_k = \varepsilon_0\theta^k$, 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 $O(\frac{1}{\varepsilon_N(1-\alpha)})$. This improves on the method of computing the N diffusions separately, which is bounded by $O\big(\frac{1}{\varepsilon_N(1-\alpha)(1-\theta)}(1-\theta^{N+1})\big)$. 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_N is exactly the same as the PUSH-work performed in computing an ε_N 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 $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$. 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 $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$. □

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

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

Table 1. *Datasets*

Graph	$\|V\|$	$\|E\|$	d_{ave}
itdk0304	190,914	607,610	6.37
dblp	226,413	716,460	6.33
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	49,514,271	18.5
hollywood	1,139,905	56,375,711	98.9
twitter	41,652,230	2,041,892,992	98
friendster	65,608,366	1,806,067,135	55.1

proving the bound on the PUSH-work. For our choices $\varepsilon_0 = 10^{-1}$, $\varepsilon_N = 10^{-6}/3$, 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 $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$, 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 non-zeros in the solution vector is bounded by $O(\varepsilon_N^{-1}(1-\alpha)^{-1})$, and so the sorting work is $O(\varepsilon_N^{-1}(1-\alpha)^{-1}\log(\varepsilon_N^{-1}(1-\alpha)^{-1}))$. This implies that sorting is the dominant subroutine of the algorithm. In practice, the bound on the number of non-zeros in the solution is loose, and the PUSH operations comprise most of the labour.

7 Experimental results on finding small conductance sets

We have presented two frameworks for computing a single personalized PageRank diffusion across multiple parameter settings. Here, we analyse their performance on a set of real-world social and information networks with varying sizes and edge-densities with the goal of identifying sets of small conductance. All datasets were altered to be symmetric and have 0s on their diagonals; this is done by deleting any self-edges and making all directed edges undirected. In addition to versions of the Facebook dataset analysed in Section 5, we test our algorithms on graphs including twitter-2010 from [20], friendster and youtube from [25,33], dblp-2010 and hollywood-2009 in [3,4], idk0304 from [29], and ljournal-2008 in [7]. See Table 1 for a summary of their properties.

7.1 The effect of ρ on conductance

Our first experimental study regards the selection of the parameter ρ for finding sets of small conductance. We already established that $\rho = 0.9$ yielded qualitatively accurate solution path plots. However, for the specific problem of identifying small conductance sets, we want to understand the effect that ρ has on the best conductance that is attained. To study this, we carry out the following experiment. For each dataset, select 100 distinct seed nodes uniformly at random. For each seed node, compute the PageRank solution

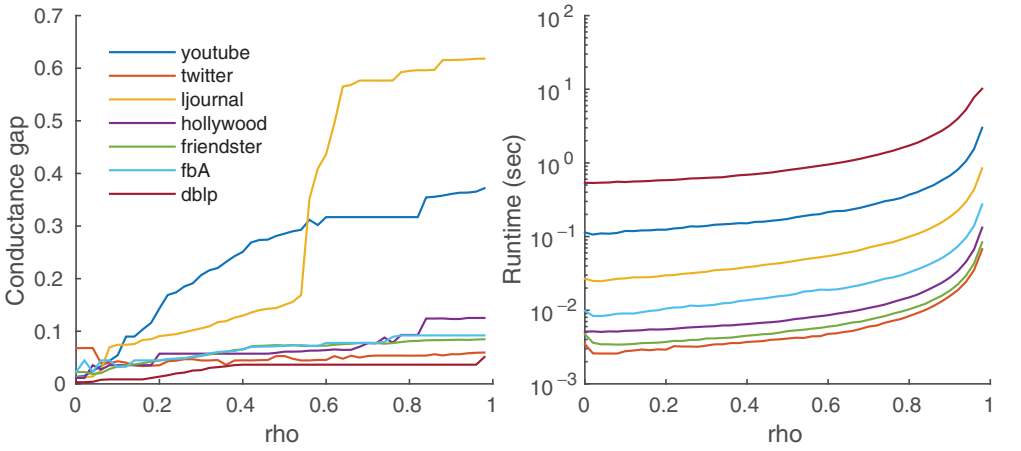

FIGURE 7. Here, we display the behaviour of the solution paths as ρ scales from 0 to 1. At left, we display the gap between $\phi(\rho)$, the best conductance found at that value of ρ, and $\phi_{\min}$, the minimum conductance found over all values of ρ. The lines depict the maximum difference, over 100 trials, of the quantity $\phi(\rho) - \phi_{\min}$. This plot shows that the best conductance found becomes *worse* as ρ approaches 1. At right, the runtime of our `ppr-path` algorithm appears to scale with $1/(1-\rho)$, which is better than the $1/(1-\rho)^2$ predicted by our theory.

paths using our algorithm `ppr-path`, with the following settings. In all cases, we use $\alpha = 0.99$ and a smallest accuracy of $\varepsilon = 10^{-5}$; we perform a different computation for each setting $\rho = 0, 0.02, 0.04, \ldots, 0.98$. This allows us to study the effect of ρ on the best conductance obtained by the solution paths. We find a curious behaviour and get the best results with small values of ρ. We will explain why this is shortly, but first we consider the results in Figure 7. In the left subplot, we report the maximum difference between the minimum conductance found for any value of ρ over the 100 trials. It can be large, for instance, 0.7 for one trial on the LiveJournal graph, where large ρ shows worse results. We remark that we choose to display the *maximum difference*, instead of the mean or median difference, because we want to understand the extent to which changing ρ can change conductance results. Changing ρ will often lead to little or no change in the conductance attained; but by displaying the maximum difference in performance, we emphasize that our new method can sometimes obtain large performance improvements.

In that same figure, we show the runtime scaling. It seems to scale with $1/(1-\rho)$, which is slightly better than expected from the bound in Theorem 1.

The greatest difference between the best conductance found for any value of ρ and the worst conductance found for any ρ occurs in the livejournal graph, with a gap of nearly 0.7. We discovered that the cause for this disparity is that large values of ρ delay the propagation of the diffusion, and so the $\rho = 0.9$ paths at $\varepsilon = 10^{-5}$ did not spread far enough to find a set of conductance near 0.07. In contrast, all paths with $\rho < 0.5$ did diffuse deep enough into the graph to identify this good conductance set. Thus, it is possible that many of the differences in conductance performance between paths with different values of ρ might in fact be caused by the *size* of the region to which the diffusion spreads for a given value of ε. Figure 8 illustrates this finding.

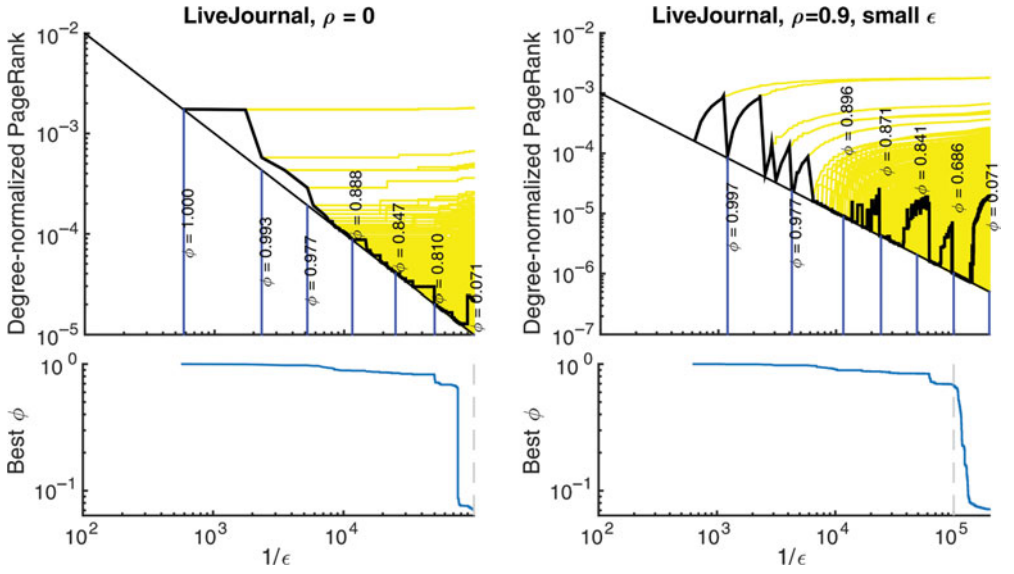

FIGURE 8. At left, the $\rho = 0$ paths identify mostly poor conductance sets $\phi \approx 0.8$, and locate a set of low conductance, $\phi = 0.0788$, only towards the end of the diffusion (the deep dip in conductance just before $\varepsilon = 10^{-5}$, indicated by a vertical dashed line). At right, we see that the $\rho = 0.9$ paths cannot find this set with $\varepsilon = 10^{-5}$ (again, indicated by a vertical dashed line). With a slightly smaller accuracy ($\varepsilon = 5 \times 10^{-6}$ instead of $\varepsilon = 10^{-5}$), the diffusion is able to spread far enough to locate the good conductance set.

Our conclusion from these experiments is that, for the goal of finding sets of small conductance, we should use small values of ρ near zero. While it sometimes happens that $\rho > 0$ slightly improves conductance, this is not a reliable observation, and so for the remaining experiments on conductance, we set $\rho = 0$. (This has the helpful side effect of making it easier to compare with our `ppr-grid`.)

7.2 Runtime and conductance: `ppr-path`

Our first method, `ppr-path`, is aimed at studying how PPR diffusions vary with the parameter ε. Towards 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 naïve 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 dataset, 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`. We describe the other two algorithms below.

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 our solution paths method with a standard implementation for computing a

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 diff.") and (2) 10,000 PPR diffusions, accuracies* k^{-1} *for* $k = 1$ *to 10,000 (labelled "multi diff."). On each dataset, we selected 100 distinct nodes uniformly at random and ran the algorithms with the settings* $\alpha = 0.99$, $\varepsilon = 10^{-5}$, *and* $\rho = 0$. *Column "num* ε*" displays the median number of distinct accuracy parameters* ε *explored by our algorithm* `ppr-path`*. Columns under "Time" report 25th, 50th, and 75th percentiles of runtimes over these 100 trials. The column "*ϕ*-ratio" lists the largest (best) ratio of conductance achieved by a single diffusion with conductance achieved by our* `ppr-path`*, showing our method can improve on the conductance found by a single diffusion by as much as a factor of* 2.09

		Single diff. time (sec.)			`ppr-path` time (sec.)			multi diff. time (sec.)			
Data	num ε	25	50	75	25	50	75	25	50	75	ϕ-ratio
itdk0304	5292	0.02	0.02	0.03	0.28	0.41	0.69	70.8	94.2	123.2	1.77
dblp	8138	0.02	0.02	0.02	0.40	0.51	0.65	87.3	97.9	111.5	1.12
youtube	2844	0.01	0.01	0.01	0.05	0.10	0.15	28.6	38.7	49.2	1.47
fb-one	3464	0.01	0.01	0.01	0.03	0.05	0.07	28.1	34.6	40.5	1.09
fbA	862	< 0.01	< 0.01	0.01	0.01	0.01	0.01	14.0	16.5	19.5	1.16
ljournal	2799	0.01	0.01	0.01	0.01	0.02	0.05	24.5	30.9	43.6	2.09
hollywood	423	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	14.0	17.2	22.4	1.19
twitter	172	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	6.5	10.3	18.1	1.05
friendster	402	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	11.1	13.6	16.6	1.09

single PPR diffusion ("Single diff." in Table 2). Columns 3 to 5 in the table give the 25th, 50th, and 75th percentiles of runtime (computed over 100 trials) for the single diffusion algorithm. To compare with our `ppr-path`, columns 6 to 8 give the 25th, 50th, and 75th percentiles (computed over 100 trials) of the runtime of our `ppr-path`. Although `ppr-path` is slower on the small 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 12 (ϕ-ratio) gives the best ratio of conductance found by `ppr-path` compared to that found by "Single diff". This shows that the solution paths can improve on the conductance obtained by a single diffusion by as much as a factor of 2.09.

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 $\varepsilon = k^{-1}$ for $k = 1$ to 10,000. This algorithm is "multi diff." in Table 2, and is essentially a naïve method for approximating the solution paths. Columns 9 to 11 give the 25th, 50th, and 75th percentiles (computed over 100 trials) of the runtimes of "multi diff.", and show that this naïve approach to computing diffusions with multiple accuracies is prohibitively slow—it is hundreds to thousands of times slower than our `ppr-path` method.

Lastly, we acknowledge here that both variations on the PPR diffusion are naïve 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 competitive baseline.

7.3 Runtime and conductance: ppr-grid

We compare our second method ppr-grid with a method called ppr-grow. The ppr-grow method is an optimized implementation [14] of the Andersen–Chung–Lang PUSH algorithm described in Section 4. Each 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 $j = 0, 1, \ldots, 4$, 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_N, the laxest and strictest accuracies (respectively), and θ, which determines the fineness of the grid of accuracy parameters. We use the settings $\alpha = 0.99$, $\varepsilon_0 = 10^{-1}$, and $\varepsilon_N = 10^{-6}/3$, and use values of θ corresponding to $N = 32$, 64, and $1,256$ different accuracy parameters.

We emphasize that this comparison with the ppr-grow method is not as naïve 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. This means that making only a single call would save little work, and would sacrifice 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, namely avoiding re-doing PUSH work between diffusion computations for different values of ε.

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-grid with ppr-grow 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 $1,256$ 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. On almost all datasets, 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. Then, columns 6 through 9 of Table 3 display how quickly ppr-grid can compute even more diffusions: whereas ppr-grow takes around 1 second to compute and analyse $N = 32$ diffusions, ppr-grid takes little more than half that time to compute on $N = 64$ diffusions (columns 6 and 7). Columns 8 and 9 show

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 25th and 75th percentile runtimes for* `ppr-grow` *(in seconds). The other columns display the median over the 100 trials of the* ratios *of the runtimes of* `ppr-grid` *(using the indicated parameter setting) with the runtime of* `ppr-grow` *on the same node. These results demonstrate that our algorithm computing over* $N = 32$ *accuracy parameters* ε_k *achieves the factor of 2 to 3 speed-up predicted by our theory in Section 6.3*

	Time (sec.) `ppr-grow`		Time ratio `grid`, $N = 32$		Time ratio `grid`, $N = 64$		Time ratio `grid`, $N = 1256$	
Data	25	75	25	75	25	75	25	75
itdk0304	6.23	8.73	0.56	0.61	0.61	0.66	1.10	1.20
dblp	4.52	7.21	0.56	0.62	0.62	0.67	1.28	1.43
youtube	1.73	2.39	0.39	0.50	0.54	0.65	3.35	4.38
fb-one	1.25	1.60	0.33	0.39	0.45	0.53	3.72	4.38
fbA	0.49	0.65	0.47	0.55	0.63	0.72	5.99	6.59
ljournal	0.82	1.20	0.44	0.55	0.58	0.74	4.57	6.12
hollywood	0.28	0.64	0.34	0.49	0.44	0.60	3.47	5.00
twitter	0.13	0.37	0.39	0.44	0.54	0.60	4.61	5.44
friendster	0.34	0.49	0.39	0.44	0.51	0.58	3.90	4.32

that `ppr-grid` can compute and analyse $N = 1256$ diffusions, nearly 40 times as many as `ppr-grow`, in an amount of time only 1.10 to 6.59 times greater than the time required by `ppr-grow`.

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 to study the meso-scale structure. 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.

8 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

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", the conductances found by* `ppr-grow` *are* 18% *larger than those found by* `ppr-grid` *with* $N = 1256$ *accuracy settings—and that comparison is on the quartile of trials where* `ppr-grid` *compares the* worst *to* `ppr-grow`*. We report the ratios in this manner (rather than their reciprocals) because in this form the values displayed are greater than 1, which distinguishes the values from conductance scores (which are between 0 and 1)*

		$N = 32$		$N = 64$		$N = 1,256$	
Data	`grow`	25	75	25	75	25	75
itdk0304	0.06	1.00	1.00	1.00	1.01	1.00	1.02
dblp	0.07	1.00	1.00	1.00	1.00	1.00	1.01
youtube	0.18	1.01	1.30	1.09	1.50	1.21	1.72
fb-one	0.37	1.06	1.16	1.10	1.26	1.18	1.37
fbA	0.56	1.00	1.05	1.00	1.06	1.00	1.09
ljournal	0.32	1.00	1.01	1.00	1.01	1.00	1.01
hollywood	0.29	1.00	1.01	1.00	1.01	1.00	1.02
twitter	0.80	1.00	1.00	1.00	1.00	1.00	1.00
friendster	0.85	1.00	1.00	1.00	1.00	1.00	1.01

the associated degree-normalized PageRank solution path plot, in order to reveal new insights about regions at different size-scales 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.

The seeded PageRank solution plots, in particular, are effective at identifying a number of subtle structures that emerge as a diffusion propagates from a set of seed nodes to the remainder of the network. We hope that these become useful tools to diagnose and study the properties of large networks.

As recently established by Ghosh et al. [10], there are many related diffusion methods that all share Cheeger-like inequalities for specific definitions of conductance. We anticipate that our solution path 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 [19]; we anticipate only mild difficulty in adapting our results to that diffusion.

Fast access to the solution path trajectories provides a number of additional opportunities that we have not yet explored. We may be able to track multiple clusters directly by managing intermediate data. We may be able to find near-optimal conductance sets that are larger than those that directly optimize the objective. Also, nodes in an egonet or larger set could be further clustered by properties of their solution paths instead of their connectivity patterns.

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.

References

[1] Andersen, R. & Chung, F. (2007) Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In Jin-Yi Cai, S. Barry Cooper, and Hong Zhu (Eds). *Theory and Applications of Models of Computation*, Springer-verlag, Berlin Heidelberg, pp. 1–12.

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

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

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

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

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

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

[8] Delvenne, J.-C., Yaliraki, S. N. & Barahona, M. (June 2010) Stability of graph communities across time scales. *Proc. Natl. Acad. Sci.* **107**(29), 12755–12760.

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

[10] Ghosh, R., Teng, S.-H., Lerman, K. & Yan, X. (2014) The interplay between dynamics and networks: Centrality, communities, and cheeger inequality. In: *KDD*, pp. 1406–1415.

[11] Gleich, D. F. (August 2015) PageRank beyond the web. *SIAM Rev.* **57**(3), 321–363.

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

[13] Gleich, D. F. & Mahoney, M. W. (2015) Using local spectral methods to robustify graph-based learning algorithms. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15*, New York, NY, USA, ACM, pp. 359–368.

[14] Gleich, D. F. & Seshadhri, C. (2012) Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In: *KDD*, pp. 597–605.

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

[16] Hastie, T., Tibshirani, R. & Friedman, J. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York, Springer.

[17] Hocking, T., Vert, J.-P., Joulin, A. & Bach, F. R. (2011) Clusterpath: An algorithm for clustering using convex fusion penalties. In: *ICML*, pp. 745–752.

[18] Jeub, L. G. S., Balachandran, P., Porter, M. A., Mucha, P. J. & Mahoney, M. W. (January 2015) Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Phys. Rev. E* **91**, 012821.

[19] Kloster, K. & Gleich, D. F. (2014) Heat kernel based community detection. In: *KDD*, pp. 1386–1395.
[20] Kwak, H., Lee, C., Park, H. & Moon, S. (2010) What is Twitter, a social network or a news media? In: *WWW '10: Proceedings of the 19th International Conference on World Wide Web*, New York, NY, USA, ACM, pp. 591–600.
[21] Langville, A. N. & Meyer, C. D. (2006) *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton, NJ, Princeton University Press.
[22] Leskovec, J., Lang, K. J., Dasgupta, A. & Mahoney, M. W. (September 2009) Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.* **6**(1), 29–123.
[23] Lindsten, F., Ohlsson, H. & Ljung, L. (2011) *Just Relax and Come Clustering! A Convexification of k-Means Clustering*, Technical Report, Linköpings Universitet.
[24] Mahoney, M. W., Orecchia, L. & Vishnoi, N. K. (August 2012) A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *J. Mach. Learn. Res* **13**, 2339–2365.
[25] Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P. & Bhattacharjee, B. (2007) Measurement and analysis of online social networks. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, IMC '07*, New York, NY, USA, ACM, pp. 29–42.
[26] Newman, M. E. J. (September 2006) Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* **74**(3), 036104.
[27] Poole, K. T. (2011) Vote View. URL: `http://voteview.com/`. [Accessed 11/10/2015].
[28] Schaeffer, S. E. (2007) Graph clustering. *Comput. Sci. Rev.* **1**(1), 27–64.
[29] C. (The Cooperative Association for Internet Data Analyais). (2005) Network datasets. Accessed 2005. URL: `http://www.caida.org/tools/measurement/skitter/router_topology/`
[30] Whang, J. J., Gleich, D. F. & Dhillon, I. S. (2013) Overlapping community detection using seed set expansion. In: *CIKM*, pp. 2099–2108.
[31] Wilson, C., Boe, B., Sala, A., Puttaswamy, K. P. & Zhao, B. Y. (2009) User interactions in social networks and their implications. In: *EuroSys*, pp. 205–218.
[32] Xie, J., Kelley, S. & Szymanski, B. K. (2013) Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.* **45**(4), 43:1–43:35.
[33] Yang, J. & Leskovec, J. (December 2012) Defining and evaluating network communities based on ground-truth. In: *IEEE 12th International Conference on Data Mining (ICDM)*, pp. 745–754.
[34] Zhou, D., Bousquet, O., Lal, T. N., Weston, J. & Schölkopf, B. (2003) Learning with local and global consistency. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 16, pp. 321–328.

Appendix

In this appendix, we provide background information for data structures and sorting operations, and give some of the more tedious details of our runtime analysis and shelf system.

Heaps, queues, and sorting

A **queue** is a data structure that enables so-called "first in, first out" access to stored entries. New elements are added at the "back" of a queue much like a queue of people; then elements are accessed only from the front of the queue. Inserting an element at the back (a process called "pushing", not to be confused with the PUSH diffusion operation

discussed in the main text), accessing the front entry, and removing the front entry (a process called "popping") are all constant-time operations.

A **max-heap** (abbreviated as just "heap" when the context is clear) is a data structure that enables constant-time access to and removal of the largest entry. A heap also enables the ability to update any existing entry or insert a new entry in $O(\log n)$ time, where n is the number of elements in the heap. A heap organizes its elements as a rooted, binary tree so that the element stored at the root is always the largest, and every node in the tree is larger than any children nodes that it is. Any time an element is updated or a new element is inserted the elements must be "re-heaped", which simply means that each node in the tree must check that it is still greater than its children.

The sorting process **bubble-sort** is a simple method for ensuring that a sorted array remains sorted after an element is updated or a new element is inserted. Assume the array is sorted in decreasing order. Then, if an element is updated (or inserted), to ensure that the array is still properly ordered, the updated (or inserted) element is compared to the element above and below it in the array. If it is larger than the element above it, the two elements are swapped (if it is smaller than the element below, then those two are swapped instead). This is repeated until no swapping is required, at which point the array is once again properly sorted.

The **bucket-sort** procedure is an approximate sorting routine that operates by putting entries into "buckets" such that the buckets are sorted, but entries within a particular bucket are not necessarily sorted. Our shelf system (presented in Section 6.3) is an example of a bucket-sort like procedure.

Completing the proof of Theorem 1

In the main text, we did not finish the proof of Theorem 1. In particular, we completed the proof of a bound on the amount of work required to perform all of the PUSH operations, but did omitted a proof of bounds on the work required to sort and sweep over the solution vectors. We present the rest of the proof here. First, we restate Theorem 1 for some context for the remainder of the proof. For additional information about the algorithm, consult Section 6.2.

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

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($\mathbf{r}$)	$\log(\lvert Q\rvert)$	$\log(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)})$
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 $Q(\mathbf{r})$	$d_j \log(\lvert Q\rvert)$	$d_j \log(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)})$

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 non-zero entries in the residual. We can upperbound this number using the total number of pushes performed (since a non-zero in the residual can exist only via a PUSH operation placing it there). We bound $|Q|$ by $O(\varepsilon_{\min}^{-1}(1-\alpha)^{-1}(1-\rho)^{-1})$, then. We remark that this is quite loose, as values of ρ near 1 actually force the solution and residual to be *sparser*, so the heap size should still be bounded by $O(\varepsilon_{\min}^{-1}(1-\alpha)^{-1})$, though we do not yet have a proof of this.

Re-sorting the solution vector via a bubblesort can involve no more operations than the length of the solution vector. Since a non-zero in entry $\mathbf{y}_j$ can exist only if a step of the algorithm operates on an entry $\mathbf{r}_j$, the number of non-zeros in $\mathbf{y}$ is bounded by the number of steps of the algorithm, i.e. $|L| \leqslant T$. 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 neighbours 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 $O(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)} \log(|Q|) + T \cdot |L|)$. * ** Finally, substituting in our loose upperbounds for T, $|Q|$, and $|L|$ mentioned above completes the proof:

$$O\left(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)} \log(\frac{1}{\varepsilon_{\min}(1-\alpha)(1-\rho)}) + \frac{1}{\varepsilon_{\min}^2(1-\alpha)^2(1-\rho)^2}\right) \leqslant O\left(\frac{1}{\varepsilon_{\min}^2(1-\alpha)^2(1-\rho)^2}\right).$$

Shelf details

Here, we present further details on how the shelf system in Section 6.3 works.

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 $\varepsilon_{k-1} > r \geqslant \varepsilon_k$ to be placed on shelf k. If $r \geqslant \varepsilon_0$, then we place r into shelf 0. Otherwise, making the substitution $\varepsilon_k = \varepsilon_0\theta^k$ and performing some algebra yields

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

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 (6) moves an entry to a shelf $l < k$. By checking for this event during the update of each individual residual entry in step (6), 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 $O(N)$.