Sublinear Estimation of a Single Element in Sparse Linear Systems

Nitin Shyamkumar, Siddhartha Banerjee, Peter Lofgren

***Abstract*— We present a bidirectional algorithm for estimating a single element in the solution of a linear system $Ax = b$, with sublinear average-case running time guarantees for sparse systems. Our work combines the von Neumann-Ulam scheme for solving linear systems with recent developments in bidirectional algorithms for estimating random-walk metrics. In particular, given a target additive-error threshold, we show how to combine a reverse local-variational technique with forward MCMC sampling such that the resulting algorithm is order-wise faster than each individual approach.**

I. Introduction

In this work, we develop a *bidirectional algorithm* for estimating a *single element* of the product of a matrix power and a vector. Formally, given any matrix $A \in \mathbb{R}^{n\times n}$, vector $\mathbf{z}$ and *a target index* $t \in \{1, 2, \dots, n\}$, we consider the problem of computing $\mathbf{p}^{\ell}_{\mathbf{z}}[t] := \langle A^{\ell}\mathbf{z}, \mathbf{e}_t\rangle$ for given exponent $\ell \in \mathbb{N}$.

Computing $A^{\ell}\mathbf{z}$ is a basic problem in matrix computations, and there is a large body of work on efficient ways of performing exact computation. However, in large-scale settings (i.e., with very large values of n), direct computation is often infeasible, and one needs to resort to estimating $A^{\ell}\mathbf{z}$. Moreover, we are interested in developing techniques which can estimate a single element of $A^{\ell}\mathbf{z}$ in running time which is provably faster than computing the entire vector.

The problem of estimating $\mathbf{p}^{\ell}_{\mathbf{z}}[t]$ has gained attention recently in the context of estimating random-walk transition probabilities, in particular, for PageRank and Personalized PageRank [**?**], and other network centrality metrics. A recent line of work [**?**], [**?**], [**?**] has shown how to develop fast bidirectional algorithms for this problem, and more generally, for settings where A^T is a *stochastic* matrix and $\mathbf{z}$ is a probability distribution (i.e., an element of the n-dimensional simplex). Our work generalizes these techniques to more general $A, \mathbf{z}$; in particular, our main result can be paraphrased as follows:

Proposition 1: Given matrix A (with $||A||_1 \le 1$), vector z (with $||z||_1 \le 1$) and any index t in $[n]$, Algorithm `BIDIR-MATRIX-POWER` (cf. Section **??**) returns estimate $\widehat{\mathbf{p}}^{\ell}_{\mathbf{z}}[t]$ such that with high probability $|\mathbf{p}^{\ell}_{\mathbf{z}}[t] - \widehat{\mathbf{p}}^{\ell}_{\mathbf{z}}[t]| < \max\{\epsilon\mathbf{p}^{\ell}_{\mathbf{z}}[t], 1/n\}$, with average running time of $\widetilde{O}\left((nnz(A)/\epsilon)^{2/3}\right)$ for uniform random choice of t.

This result is interesting in that it generalizes the existing results for stochastic matrices to arbitrary matrices, albeit with a loss in the running time scaling [1]. Moreover, computing $\left(A^{\ell}\mathbf{z}\right)[t]$ is often a subroutine in more complex algorithms for various applications. In Section **??**, we focus on one such application – the von Neumann-Ulam scheme for solving linear equations [**?**]. This has been gaining interest as a promising candidate for solving large-scale equations, owing to its ease of parallelization and asynchronous and local nature [**?**], [**?**], [**?**]. More generally however, this algorithm may prove useful in more complex estimation tasks which use matrix polynomials as approximators.

A. Related Work

Previous work has addressed both the problem of estimating a single component of a matrix equation and the problem of computing matrix inversions through Monte Carlo methods. The von Neumann-Ulam algorithm shows that the inverse of B by defining $A = I - B$ and running random walks over the induced graph of A [**?**]. As long as the spectral norm of A is less than 1, the expectation of these random walks is exactly $(A)_{ij}$ where i is the start node and j is the end node. With these methods, one can solve the system $\mathbf{x} = G\mathbf{x} + \mathbf{z}$ provided the spectral norm $\rho(G) < 1$ since $\mathbf{x} = (I - G)^{-1}\mathbf{z}$, exactly the problem the von Neumann-Ulam algorithm solves. [**?**] proved that there exist matrices G satisfying $\rho(G) < 1$ but with $||G||_\infty > 1$, and that convergence of the von Neumann-Ulam algorithm is not guaranteed on this class of matrices. Wu and Gleich address this issue by proposing a new algorithm that will converge as long as $\rho(G^+) < 1$, a weaker condition than $\rho(G) < 1$ [**?**].

We can also view the problem of estimating a single component $\mathbf{x}[t]$ as computing the contributions in reverse from a source distribution to $\mathbf{x}[t]$.This problem was originally addressed for the special case of computing Personalized Page Rank (PPR) in [**?**] via the Reverse-Push algorithm, which pushes back the unit contribution using a series of reverse operations. [**?**] shows that this problem for the specific case of PPR can return an ϵ accurate estimate of $ppr(v)$ assuming $ppr(v) \ge \alpha$ in time $O\left(\frac{1}{\alpha\epsilon}\right)$. This approach also resembles the work in [**?**], which describes a similar algorithm using an asynchronous procedure. [**?**] performs a series of local updates using a residual vector, and show that with this procedure one can estimate $\mathbf{x}[t]$ satisfying $|\mathbf{x}[t] - \hat{\mathbf{x}[t]}| \le \epsilon||\mathbf{x}||$ in time $O\left(\min\{\epsilon^{\ln d/\ln(||G||_2)}, n\ln\epsilon/\ln(||G||_2)\}\right)$.

Our work is based on recently developments on bidirectional algorithms for estimating single-state transition

N. Shyamkumar is with the Department of Computer Science, and S. Banerjee is with the School of Operations Research and Information Engineering, Cornell University, USA, Email: `nhs56@cornell.edu`, `sbanerjee@cornell.edu`. P. Lofgren is with the Department of Computer Science, Stanford University, USA, Email: `plofgren@cs.stanford.edu`.

[1]For estimating ℓ-step transition probabilities, the running time of an equivalent algorithm in [**?**] is $\widetilde{O}\left((nnz(A)/\epsilon)^{1/2}\right)$ for uniform random choice of t.

probabilities in Markov chains. Such algorithms were first developed for *reversible Markov chains* using random-walk collision statistics; in particular, Kale et al. [?] proposed such a technique for estimating length-2ℓ random walk transition probabilities in a *regular undirected graph*. The main idea is that to test if a random walk goes from s to t in 2ℓ steps with probability $\geq \delta$, we can generate two independent random walks of length ℓ, starting from s and t respectively, and detect if they collide, i.e., terminate at the same intermediate node. The critical observation is that using $\sqrt{n}$ walks from s and t gives n potential collisions, which is sufficient to estimate probabilities on the order of $1/n$. This argument draws from older ideas on using the *birthday-paradox* in estimation problems [?]. A bidirectional algorithm for general graphs was first developed by Lofgren et al. [?] for PageRank estimation; the argument was subsequently simplified in [?], and extended to general Markov chains in [?]. Our work here further generalizes this line of work to computing single elements of powers of arbitrary matrices, with a particular application to solving for single elements in sparse linear systems.

II. Solving Linear Equations via Series Approximation

Unless specified otherwise, we use boldface letters (e.g. $\mathbf{x}, \mathbf{y}$) to denote vectors in $\mathbb{R}^{n\times 1}$, and capital letters (e.g. A, Q) to denote matrices in $\mathbb{R}^{n\times n}$.

Given a linear system $\mathbf{y} = A\mathbf{x}$, where A is a positive definite matrix, our aim is to estimate $\mathbf{x}[t]$ for some given index $t \in [n]$. This can be done by directly solving for $\mathbf{x}$; however, we are interested in settings where n is very large, and hence direct solution techniques may be impractical.

One approach for approximating the solution to $\mathbf{y} = A\mathbf{x}$ is to expand it via the Neumann series and then compute the leading terms of the summation. In particular, if A is positive definite, we can find γ such that $G = I - \gamma A$ satisfies $\rho(G) < 1$. Then $\gamma \mathbf{y} = (I - (I - \gamma A)\mathbf{x}$ and we have a new system $\mathbf{x} = G\mathbf{x} + \mathbf{z}$ where $\mathbf{z} = \gamma \mathbf{y}$ and $G = I - \gamma A$. Since we ensure $\rho(G) < 1$, we can write $\mathbf{x}$ as a von Neumann series: $\mathbf{x} = \sum_{k=0}^{\infty} G^k \mathbf{z}$. Thus to find the t component of the solution vector $\mathbf{x}$, or $\mathbf{x}[t]$, we perform the operation $\langle \mathbf{x}, \mathbf{e}_t \rangle = \sum_{k=0}^{\infty} \langle G^k \mathbf{z}, \mathbf{e}_t \rangle$. Similar transformations have been used in [?], [?], [?].

Computing $\mathbf{x}[t]$ then amounts to computing $\mathbf{p}_{\mathbf{z}}^{\ell}[t] := \langle G^{\ell}\mathbf{z}, \mathbf{e}_t \rangle = \langle \mathbf{z}, (G^T)^{\ell} \mathbf{e}_t \rangle$ for any ℓ and taking their sum for some $\ell \in \{0, \ldots, \ell_{\max}\}$ where $\ell_{\max}$ is a finite term truncating the power series. Let $Q := G^T$; prior work by Banerjee and Lofgren [?] shows how to compute $\mathbf{p}_{\mathbf{z}}^{\ell}[t]$ for the special case where $\mathbf{z}$ is a probability vector and Q is a *stochastic matrix* (i.e., with all nonnegative entries, and each row summing to 1). We now extend this result for any $\mathbf{z}$ and a special class of matrices Q.

Note that the error from truncating the series to $\ell_{\max}$ can be determined a priori; using bounds for $\mathbf{z}$ and the condition that G has spectral norm less than 1, we can set $\ell_{\max} \geq \frac{1}{\ln \rho(G)} \ln \left(\frac{\Delta(1-\rho(G))}{||\mathbf{z}||} \right)$ to bound the series truncation error by Δ.

Proof: Let ε be the error from truncating the power series to $\ell_{\max}$. Then by definition:

$$\varepsilon = \left| \sum_{\ell=0}^{\infty} \langle \mathbf{z}, Q^{\ell} \mathbf{e}_t \rangle - \sum_{\ell=0}^{\ell_{\max}} \langle \mathbf{z}, Q^{\ell} \mathbf{e}_t \rangle \right|$$
$$= \left| \left\langle \mathbf{z}, Q^{\ell_{\max}+1} \sum_{\ell=0}^{\infty} Q^{\ell} \mathbf{e}_t \right\rangle \right|$$

For any inner product of the form $\langle \mathbf{z}, Q^{\ell} \mathbf{e}_t \rangle$, by Cauchy - Schwarz we have $|\langle \mathbf{z}, Q^{\ell} \mathbf{e}_t \rangle| \leq ||\mathbf{z}|| \cdot ||Q^{\ell} \mathbf{e}_t|| \leq ||\mathbf{z}|| \rho(Q)^{\ell}$. Hence we have $\varepsilon \leq ||\mathbf{z}|| \rho(Q)^{\ell_{\max}+1} \left(\frac{1}{1-\rho(Q)} \right)$ Now suppose we want our error $\epsilon \leq \Delta$. Then we can set the upper bound for ϵ to Δ and solve for $\ell_{\max}$. Thus, if we take $\ell_{\max} \geq \frac{1}{\ln \rho(Q)} \ln \left(\frac{\Delta(1-\rho(Q))}{||\mathbf{z}||} \right)$, we will have an error of at most Δ when approximating the series. ■

Since we can a priori bound the error resulting from truncating the von Neumann series to $\ell_{\max}$, we will focus on the problem of estimating $\mathbf{p}_{\mathbf{z}}^{\ell}[t]$. It is then straightforward to develop overall error bounds by combining the results of the additive truncation error Δ and the relative error guarantees we prove for our bidirectional algorithm in Theorem 1.

III. Existing Algorithms for Estimating Matrix Powers

As mentioned above, our main contribution in this work is to develop a bidirectional algorithm for estimating $\mathbf{p}_{\mathbf{z}}^{\ell}[t] = \langle \mathbf{z}, Q^{\ell} \mathbf{e}_t \rangle$. To do so, we first describe two existing algorithms, which we use as primitives for our procedure – a forward MCMC technique based on the von Neumann-Ulam scheme [?], [?], and a local variational method proposed by Andersen et al. [?] for computing PageRank, and used by Lee et al. [?] in this setting. We present these along with statements of their correctness and running time – some of these results follow directly from previous work (as we note in the appropriate sections), and their proofs are included here mainly for the sake of completeness.

We first introduce some notation which will help us better describe the algorithm. Drawing parallels to the case where Q is a stochastic matrix and $\mathbf{z}$ an element in the n-dimensional simplex (as in [?]), we define a (weighted) directed graph $\mathcal{G}_Q(\mathcal{V}, \mathcal{E})$ with states $\mathcal{V} = [n]$, and edges $(i,j) \in \mathcal{E}$ if $Q_{ij} \neq 0$. Each edge $(i,j) \in \mathcal{E}$ has an associated weight $w_{ij} \in \mathbb{R}$, which we describe later. We refer to the label for a node $v \in V$ (i.e., a dimension $v \in [n]$) as a *dimension-index*, and the exponent of Q as the *step-index*. We also use $\mathbf{e}_v$ denote the indicator for index v (i.e., $\mathbf{e}_v[i] = \mathbb{1}_{i=v}$). Finally, we define Q^+ to denote the matrix with $Q^+_{ij} = |Q_{ij}|$. Many of our bounds depend on the norm $||Q^+||_{\infty} = ||Q||_{\infty}$, i.e., the maximum (absolute) row-sum of Q; for ease of notation, we henceforth define $\beta = ||Q||_{\infty}$.

A. Computing Matrix Powers via Iterative Local-Update

One approach towards estimating $\mathbf{p}_{\mathbf{z}}^{\ell}[t]$ is via a standard power iteration for computing $Q^{\ell}\mathbf{e}_t$. In settings where n is large enough such that a direct power iteration is infeasible, one can use a 'local' power iteration, which essentially corresponds to a natural dynamic programming update. Informally, the algorithm estimates $\mathbf{p}^{\ell}_{\mathbf{z}}[t]$ by starting off with a mass of 1 on dimension-index t, and then 'pushing' this mass in reverse along the edges of graph G_Q.

To describe this REVERSE-LOCAL-UPDATE algorithm, we first define a REVERSE-PUSH operation corresponding to the DP iteration. This is directly adapted from the algorithm in Andersen [?] for the personal PageRank problem (and more generally for a stochastic matrix Q); however, it is straightforward to show that the invariant that holds for any matrix Q (in fact, it does not require Q to be full rank, as we have assumed).

The REVERSE-PUSH operation is a standard dynamic programming iteration which is used to compute an estimate $\widehat{\mathbf{p}}^{\ell}_{\mathbf{z}}[t]$ by proceeding 'in reverse' from t. Essentially,REVERSE-PUSH is a local power-iteration for computing $Q^{\ell}\mathbf{e}_t$; instead of performing a full power-iteration, it adaptively exploits any sparsity in the computation. This operation was defined in the form given below in [?], and subsequently used as a primitive in [?], [?].

For each step-index $k \in \{0, 1, \dots, \ell\}$, we store two vectors: the *estimate vector* $\mathbf{q}^k_t$ and the *residual vector* $\mathbf{r}^k_t$. We initialize all $\mathbf{r}^k_t, \mathbf{q}^k_t, k \in [\ell]$ to 0, except for $\mathbf{r}^0_t$, which we set to $\mathbf{e}_t$. Now, given any dimension-index $v \in [n]$ and step-index $k \in [\ell]$, the REVERSE-PUSH operation iteratively updates these vectors as follows:

Algorithm 1 REVERSE-PUSH(t, v, k)

```
Inputs: Matrix $Q$, estimates $\mathbf{q}^k_t$, residuals $\mathbf{r}^k_t, \mathbf{r}^{k+1}_t$
1: return New estimates $\widetilde{\mathbf{q}}^k_t$ and residuals $\widetilde{\mathbf{r}}^k_t$ computed as:
```

$$\begin{aligned}\widetilde{\mathbf{q}}^k_t &\leftarrow \mathbf{q}^k_t + \left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle \mathbf{e}_v \\ \widetilde{\mathbf{r}}^k_t &\leftarrow \mathbf{r}^k_t - \left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle \mathbf{e}_v \\ \widetilde{\mathbf{r}}^{k+1}_t &\leftarrow \mathbf{r}^{k+1}_t + \left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle (Q\mathbf{e}_v)\end{aligned}$$

The REVERSE-PUSH iteration results in the following critical invariant for the estimate and residual vectors:

Lemma 1: Given the initialization described above, after any sequence of REVERSE-PUSH operations, and for any $\mathbf{z} \in \mathbb{R}^n$ and $\ell \geq 0$, the estimates $\{\mathbf{q}^k_t\}$ and residuals $\{\mathbf{r}^k_t\}$ satisfy the following invariant:

$$\mathbf{p}^{\ell}_{\mathbf{z}}[t] = \left\langle \mathbf{z}, \mathbf{q}^{\ell}_t \right\rangle + \sum_{k=0}^{\ell} \left\langle \mathbf{z}, Q^k \mathbf{r}^{\ell-k}_t \right\rangle = \left\langle \mathbf{z}, \mathbf{q}^{\ell}_t + \sum_{k=0}^{\ell} Q^k \mathbf{r}^{\ell-k}_t \right\rangle$$

The above invariant was first stated in [?] for the case of PageRank vectors. For the sake of completeness, we present a proof below for general matrices, adapted from [?]; an identical invariant is given in [?].

Proof: For our chosen initialization (i.e., $\mathbf{r}^0_t = \mathbf{e}_t$, and all other estimate and residual vectors set to 0), the invariant simplifies to $\mathbf{p}^{\ell}_{\mathbf{z}}[t] = \left\langle \mathbf{z}, Q^{\ell}\mathbf{e}_t \right\rangle$ which is true by definition. Now, assuming the invariant holds at any stage with vectors $\{\mathbf{q}^k_t\}, \{\mathbf{r}^k_t\}_{k\in[\ell]}$, and let $\{\widetilde{\mathbf{q}}^k_t\}, \{\widetilde{\mathbf{r}}^k_t\}_{k\in[\ell]}$ be the new vectors after executing a REVERSE-PUSH(t, v, k) operation for any given $k \in [\ell]$ and $\mathbf{z} \in \mathbb{R}^n$. We define:

$$\Delta^k_v = \left(\tilde{\mathbf{q}}^{\ell}_t + \sum_{i=0}^{\ell} (Q^i)\tilde{\mathbf{r}}^{\ell-i}_t\right) - \left(\mathbf{q}^{\ell}_t + \sum_{i=0}^{\ell} (Q^i)\mathbf{r}^{\ell-i}_t\right)$$

Now to show that the invariant holds following REVERSE-PUSH(t, v, k), it suffices to show that Δ^k_v is zero for any $v \in V$ and $k \in [\ell]$.

We now have three cases: (i) if $\ell < k$, then the REVERSE-PUSH(t, v, k) operation does not affect the residual or estimate vectors $\{\mathbf{q}^i_t, \mathbf{r}^i_t\}_{i<k}$, and hence $\Delta^k_v = 0$; (ii) If $\ell = k$, we have:

$$\begin{aligned}\Delta^k_v &= \left(\tilde{\mathbf{q}}^k_t + \tilde{\mathbf{r}}^k_t\right) - \left(\mathbf{q}^k_t + \mathbf{r}^k_t\right) \\ &= \mathbf{q}^k_t + \left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle \mathbf{e}_v + \mathbf{r}^k_t - \left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle \mathbf{e}_v - \mathbf{q}^k_t - \mathbf{r}^k_t = 0\end{aligned}$$

(iii) Finally, when $\ell > k$, we have:

$$\begin{aligned}\Delta^k_v &= Q^{\ell-k}\left(\tilde{\mathbf{r}}^k_t - \mathbf{r}^k_t\right) + Q^{\ell-k-1}\left(\tilde{\mathbf{r}}^{k+1}_t - \mathbf{r}^{k+1}_t\right) \\ &= -\left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle Q^{\ell-k}\mathbf{e}_v + \left\langle \mathbf{r}^k_t, \mathbf{e}_v \right\rangle Q^{\ell-k-1}\left(Q\mathbf{e}_v\right) = 0\end{aligned}$$

Hence we have shown that the invariant is preserved for any sequence of reverse push operations. ■

The above invariant gives a natural iterative algorithm for computing $\mathbf{p}^{\ell}_{\mathbf{z}}[t]$, by performing repeated REVERSE-PUSH operations and controlling the residual vectors $\mathbf{r}^k_t$, and using $\widehat{\mathbf{p}}^{\ell}_{\mathbf{z}}[t] = \langle \mathbf{z}, \mathbf{q}^{\ell}_t \rangle$ as the estimate. Depending on the norm we choose to control, we can get a bound for the error via Hölder's inequality. In particular, controlling the infinity norm (i.e., the maximum absolute value of the residual vectors) to be less than some chosen $\delta_r > 0$ gives us a bound: $|\mathbf{p}^{\ell}_{\mathbf{z}}[t] - \langle \mathbf{z}, \mathbf{q}^{\ell}_t \rangle| \leq ||x||_1 \delta_r \beta^{\ell}$ [2].

Algorithm 2 REVERSE-LOCAL-UPDATE(t, Q, ℓ, δ_r)

```
Inputs: Matrix $Q$, maximum step-index $\ell$, target residual threshold $\delta_r$
1: Initialize all residual $\mathbf{r}^k_t$ and estimate vectors $\mathbf{q}^k_t, k \in [\ell]$ to 0; set $\mathbf{r}^0_t = \mathbf{e}_t$
2: for  $k \in \{0, 1, 2, ...\ell\}$  do
3:     while $\exists v$ such that $\left|\mathbf{r}^k_t[v]\right| > \delta_r$ do
4:         REVERSE-PUSH$(t, v, k)$
5:     end while
6: end for
7: return  $\{\mathbf{q}^k_t\}, \{\mathbf{r}^k_t\}_{k\in[\ell]}$
```

Finally, we want to bound the running time of REVERSE-LOCAL-UPDATE(t, Q, ℓ, δ_r). It is easy to see that in the worst case, the running time can be as much as the ℓ-hop in-neighborhood of t in Q. However, for a *uniform random* choice of t, we can obtain a more informative bound. Recall we define $\beta = ||Q||_{\infty}$. Now we have the following:

Lemma 2: For any $Q \in \mathbb{R}^{n\times n}$ and uniform random dimension-index $t \in [n]$, the expected running time of

[2]This approach was used in [?], [?], [?]; an alternative is to control $||r^k_t||_2$ giving error bounds in terms of $||x||_2$, which was suggested in [?].

REVERSE-LOCAL-UPDATE(t, Q, ℓ, δ_r) is

$$O\left(\frac{nnz(Q)}{n\delta_r}(\ell+1)\beta^\ell\right)$$

In particular, note that if $||Q||_\infty \leq 1$ and $\ell = O(1)$, then the average running time is $O\left(\frac{nnz(Q)}{n\delta_r}\right)$.

Proof: Let $T(t)$ be the running time of REVERSE-LOCAL-UPDATE(t, Q, ℓ, δ_r). Recall we define Q^+ as the matrix with $Q^+_{ij} = |Q_{ij}|$; let $\hat{T}(t)$ be the running time of REVERSE-LOCAL-UPDATE(t, Q^+, ℓ, δ_r). Then we have that for every matrix Q and every t, we have $\hat{T}(t) > T(t)$ – this follows from the fact that any cancellation between positive and negative residuals in REVERSE-LOCAL-UPDATE(t, Q^+, ℓ, δ_r) can only decrease the number of iterations. Also, note that under Q^+, since all residuals are positive, we have that for any $k \leq \ell, v \in [n]$, the residuals satisfy $\mathbf{r}^k_t[v] \leq \left(\mathbf{e}^T_v Q^\ell\right)[t]$.

Now let $d_i := \sum_j \mathbb{1}_{\{Q_{ij} \neq 0\}}$, i.e., the support of i^{th} row in Q, and $\mathbf{r}^k_t$ denote the residuals under REVERSE-LOCAL-UPDATE(t, Q^+, ℓ, δ_r). Recall we define $\beta = ||Q||_\infty$.From Algorithm **??**, we have $\hat{T}(t) = \sum_{k=0}^{\ell}\sum_{v\in[n]} \mathbb{1}_{\mathbf{r}^k_t[v] > \delta_r}$. Thus, the expected running time over a uniform random choice of $t \in [n]$ is given by

$$\begin{aligned}
\frac{1}{n}\sum_t \hat{T}(t) &= \frac{1}{n}\sum_{t\in[n]}\sum_{k=0}^{\ell}\sum_{v\in[n]} \mathbb{1}_{\{\mathbf{r}^k_t > \delta_r\}} d_v \\
&= \frac{1}{n}\sum_{k=0}^{\ell}\sum_{v\in[n]}\sum_{t\in[n]} \mathbb{1}_{\{\mathbf{r}^k_t > \delta_r\}} d_v \\
&\leq \frac{1}{n}\sum_{k=0}^{\ell}\sum_{v\in[n]} \mathbb{1}_{\{(\mathbf{e}^T_w (Q^+)^k)[t] > \delta_r\}} d_v \\
&= \frac{1}{n}\sum_{k=0}^{\ell}\sum_{v\in[n]} \frac{||\mathbf{e}^T_w (Q^+)^k||_1}{\delta_r} d_v \\
&\leq \frac{1}{n}\sum_{k=0}^{\ell}\sum_{v\in[n]} \frac{||Q^+||^k_\infty}{\delta_r} d_v \\
&\leq (\ell+1)\,\beta^\ell \frac{nnz(Q)}{n\delta_r}
\end{aligned}$$

■

B. Computing Matrix Powers via MCMC Sampling

In the previous section, we computed $\mathbf{p}^\ell_\mathbf{z}[t]$ by working backwards from t. Note that our final algorithm is independent of $\mathbf{z}$. We now present an alternate technique which is based on a forward MCMC sampling technique called the von Neumann-Ulam scheme. In this case, the algorithm starts from $\mathbf{z}$, and computes $\mathbf{p}^\ell_\mathbf{z}[t]$ for all $t \in [n]$.

More generally, given any vectors $\mathbf{a}$ and $\mathbf{b}$ and matrix Q, the von Neumann-Ulam scheme can be used for computing $\langle \mathbf{a}, Q^\ell \mathbf{b}\rangle$. To understand the algorithm, note that we can expand $\langle \mathbf{a}, Q^\ell \mathbf{b}\rangle$ as the sum $\sum_{(v_0,\ldots,v_\ell)\in V^\ell}\left(\prod_{j\in[\ell]} Q_{v_{j-1}v_j}\right)\mathbf{a}[v_0]\mathbf{b}[v_\ell]$. Now we can interpret this sum as an expectation over an ℓ-step random walk $W = (V_0, V_1, \ldots, V_\ell)$ on $\mathcal{G}$, specified as follows:

- V_0, the starting node for the random walk, is sampled from $\sigma_\mathbf{a} = \{|\mathbf{a}[i]|/||\mathbf{a}||_1\}_{i\in[n]}$, and has an associated weight $w_{V_0} = sgn(\mathbf{a}[V_0])||\mathbf{a}||_1$.
- The transition probability matrix for the walk is given by $P_{ij} = \{|Q_{ij}|/||Q_i||_1\}$ (where $||Q_i||_1$ is the 1-norm of the i^{th} row of Q).
- Each edge $(i,j) \in E$ has associated weight $w_{ij} = sgn(Q_{ij})||Q_i||_1$.
- The 'score' for a walk W is the product of weights of traversed edges, i.e.,

$$S^\ell_t(W) = w_{V_0}\prod_{i=0}^{\ell-1} w_{V_iV_{i+1}}\mathbf{b}[V_\ell]$$

This procedure is summarized in Algorithm **??**.

Algorithm 3 MCMC-SAMPLER$(Q, \ell, \mathbf{a}, \mathbf{b})$

```
Inputs: Matrix $Q$, exponent $\ell$, vectors $\mathbf{a}$ and $\mathbf{b}$
1: Construct transition matrix $P_{ij} = \{|Q_{ij}|/||Q_i||_1\}$ and starting measure $\sigma_a = \{|\mathbf{a}[i]|/||\mathbf{a}||_1\}_{i\in V}$
2: Define node weights $w_{V_0} = sgn(\mathbf{a}[V_0])||\mathbf{a}||_1$ and edge weights $w_{V^iV^j} = sgn(Q_{V^iV^j})||Q_i||_1$
3: Construct source distribution $\sigma_\mathbf{a}$ with $\sigma_\mathbf{a}[i] = \frac{\mathbf{z}[i]}{||\mathbf{z}||_1}$
4: Sample $V^0 \sim \sigma_a$, and generate a random walk $W = \{V^0, V^1, \ldots, V^\ell\}$ of length $\ell$ on $\mathcal{G}$ using transition probability $P$.
5: return Walk-score $S^\ell_t(W) = w_{V_0}\prod_{i=0}^{\ell-1} w_{V_iV_{i+1}}\mathbf{b}[V_\ell]$
```

Recall we define $\beta = ||Q||_\infty$. Now we have the following Lemma.

Lemma 3: MCMC-SAMPLER$(Q, \ell, \mathbf{a}, \mathbf{b})$ returns a walk-score $S^\ell_t(W)$ which satisfies:

1) $\mathbb{E}_{W\sim\sigma_\mathbf{a}(P)^\ell}\left[S^\ell_t(W)\right] = \langle \mathbf{a}, Q^\ell \mathbf{b}\rangle$
2) $S^\ell_t(W) \in [-\beta^\ell||\mathbf{a}||_1||\mathbf{b}||_\infty,\ \beta^\ell||\mathbf{a}||_1||\mathbf{b}||_\infty]$

Proof: We can expand $\mathbb{E}_{W\sim\sigma_\mathbf{a}(P)^\ell}\left[S^\ell_t(W)\right]$ to obtain:

$$\begin{aligned}
&\mathbb{E}_{W\sim\sigma_\mathbf{a}(P)^\ell}\left[S^\ell_t(W)\right] \\
&= \sum_{(V^0,\ldots V^\ell)\in[n]^{\ell+1}}\left(\frac{sgn(\mathbf{a}[V_0])||\mathbf{a}||_1|\mathbf{a}[V^0]|}{||\mathbf{a}||_1}\right. \\
&\times\left.\left(\prod_{j\in[0,\ell-1]}\frac{sgn(Q_{V^jV^{j+1}}||Q_{V^j}||_1)|Q_{V^jV^{j+1}}|}{||Q_{V^j}||_1}\right)\mathbf{b}[V^\ell]\right) \\
&= \sum_{(V^0,\ldots V^k)\in[n]^{\ell+1}}\left(\mathbf{a}[V_0]\left(\prod_{j\in[0,\ell-1]} Q_{V^jV^{j+1}}\right)\mathbf{b}[V^\ell]\right)
\end{aligned}$$

The final expression is exactly the definition of $\langle \mathbf{a}, Q^\ell \mathbf{b}\rangle$; hence $\mathbb{E}_{W\sim\sigma_\mathbf{a}(P)^\ell}\left[S^\ell_t(W)\right] = \langle \mathbf{a}, Q^\ell \mathbf{b}\rangle$.

Next, in order to bound the score $S^\ell_t(W)$, recall that

$$S^\ell_t(W) = w_{V_0}\prod_{i=0}^{\ell-1} w_{V_iV_{i+1}}\mathbf{b}[V_\ell]$$

We defined $w_{V_0} = sgn(\mathbf{a}[V_0])||\mathbf{a}||_1$, so trivially $|w_{V_0}| \leq ||\mathbf{a}||_1$. Additionally, $w_{V_iV_{i+1}} = sgn(Q_{V^iV^j})||Q_i||_1$ and by definition $||Q_i||_1 \leq ||Q||_\infty$ and $|\mathbf{b}[V^\ell]| \leq ||\mathbf{b}||_\infty$. Hence, since we are multiplying ℓ edge scores, we finally obtain $|S^\ell_t(W)| \leq ||Q||^\ell_\infty||\mathbf{a}||_1||\mathbf{b}||_\infty$ as stated in the lemma. ■

IV. A Bidirectional Algorithm for Computing Matrix Powers

Finally, we present our main contribution: a bidirectional estimator for $\mathbf{p}^\ell_{\mathbf{z}}[t] = \langle \mathbf{z}, Q^\ell \mathbf{e}_t\rangle$. Our algorithm follows the general structure proposed by Lofgren et al. [**?**], [**?**] for PageRank and Markov Chain transition probability estimation. It comprises of two distinct components: first we use `REVERSE-LOCAL-UPDATE` to estimate approximate values of $\left(Q^\ell \mathbf{e}_t\right)[i]$ for all steps $\ell \in [\ell_{\max}]$ and $i \in [n]$. We then use `MCMC-SAMPLER` to reduce the error in these estimates to get our desired accuracy.

Intuitively, the advantage we gain from combining the two previous algorithms is that running a small amount of local-update reduces the variance in the walk-scores significantly, which then allows us to perform sampling more effectively. More specifically, given a desired error threshold δ, we first run `REVERSE-LOCAL-UPDATE`(t, Q, ℓ, δ_r) for an appropriately chosen $\delta_r \gg \delta$ (cf. Theorem **??**). At this point, from Lemma **??**, we know that we have $\mathbf{p}^\ell_{\mathbf{z}}[t] = \left\langle \mathbf{z}, \mathbf{q}^\ell_t\right\rangle + \sum_{k=0}^{\ell} \left\langle \mathbf{z}, Q^k \mathbf{r}^{\ell-k}_t\right\rangle$, with $\mathbf{r}^k_t[v] \le \delta_r$ for all $v \in [n], k \le \ell$. Now, instead of ignoring the residual terms (as done in [**?**], [**?**]; cf. Section **??**), we can further reduce our error by using `MCMC-SAMPLER`$(Q, k, \mathbf{z}, \mathbf{r}^k_t)$ to estimate the residual $\langle \mathbf{z}, \mathbf{r}^k_t\rangle$ for all $k \le \ell$. Note however that the resulting error is better than ignoring the residual, and also better than directly executing `MCMC-SAMPLER`$(Q, \ell, \mathbf{z}, \mathbf{e}_t)$, as the residuals have much smaller magnitude than 1, and hence the walk-scores have lower variance.

Algorithm 4 `BIDIR-MATRIX-POWER`$(Q, \mathbf{z}, t, \ell)$

```
Inputs: Matrix $Q$, exponent $\ell$, vector $\mathbf{z}$, target index $t$.
1: Compute estimate and residual vectors $\{\mathbf{r}^k_t, \mathbf{q}^k_t\}_{\{k\le\ell\}}$ =
   REVERSE-LOCAL-UPDATE$(t, Q, \ell_{\max}, \delta_r)$
2: for  $i \in [n_f]$ do
3:     $k \sim Unif[0, \ell]$
4:     $S^\ell_{i,t}$ = MCMC-SAMPLER $(Q, k, \mathbf{z}, \mathbf{r}^{\ell-k}_t)$
5: end for
6: return  $\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t] = \langle \mathbf{z}, \mathbf{q}^\ell_t\rangle + \frac{\ell+1}{n_f}\sum_{i=1}^{n_f} S^\ell_{i,t}$
```

First, we observe that `BIDIR-MATRIX-POWER` always returns an unbiased estimate for $\mathbf{p}^\ell_{\mathbf{z}}[t]$ for any choice of n_f

Lemma 4: `BIDIR-MATRIX-POWER` returns an unbiased estimator of $\mathbf{p}^\ell_{\mathbf{z}}[t]$, i.e. $\mathbb{E}[\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t]] = \mathbf{p}^\ell_{\mathbf{z}}[t]$

Proof: By definition of $\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t]$ and linearity of expectation

$$\mathbb{E}\left[\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t]\right] = \left\langle \mathbf{z}, \mathbf{q}^\ell_t\right\rangle + (\ell+1)\mathbb{E}_{K\sim Unif[0,\ell]}\left[S^K_t\right]$$

From Lemma **??**, we know `MCMC-SAMPLER` $(Q, k, \mathbf{z}, \mathbf{r}^k_t)$ returns an estimate satisfying $\mathbb{E}\left[S^k_t\right] = \left\langle \mathbf{z}, Q^k \mathbf{r}^k_t\right\rangle$ for all $k \le \ell$. Thus, with $K \sim Unif[0, \ell]$, we have

$$\begin{aligned}\mathbb{E}\left[\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t]\right] &= \left\langle \mathbf{z}, \mathbf{q}^\ell_t\right\rangle + (\ell+1)\mathbb{E}_K\left[\left\langle \mathbf{z}, Q^K \mathbf{r}^{\ell-K}_t\right\rangle\right] \\ &= \left\langle \mathbf{z}, \mathbf{q}^\ell_t\right\rangle + \sum_{k=0}^{\ell}\left\langle \mathbf{z}, Q^k \mathbf{r}^{\ell-k}_t\right\rangle\end{aligned}$$

However, Lemma **??** states that after any sequence of reverse push operations, we have the invariant $\mathbf{p}^\ell_{\mathbf{z}}[t] = \left\langle \mathbf{z}, \mathbf{q}^\ell_t\right\rangle + \sum_{k=0}^{\ell}\left\langle \mathbf{z}, Q^k \mathbf{r}^{\ell-k}_t\right\rangle$. Thus, $\mathbb{E}\left[\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t]\right] = \mathbf{p}^\ell_{\mathbf{z}}[t]$. ■

To choose n_f to get the desired accuracy, we need the following concentration bound, which is a restatement of Hoeffding's inequality (cf. Chapter 1 in [**?**]).

Lemma 5 (Hoeffding's Inequality): Given $\{X_i\}$ independent random variables s.t. for all i, $X_i \in [-c, c]$ a.s., and $|\mathbb{E}[X_i]| \ge a$. Then $X = \sum_{i=1}^{n} X_i$ satisfies $\mathbb{P}[|X - \mathbb{E}[X]| \ge \epsilon\mathbb{E}[X]] \le p_{fail}$ provided that

$$n \ge \frac{2c^2}{\epsilon^2 a^2}\ln\left(\frac{2}{p_{fail}}\right)$$

Now we can state our main result, where we show how to choose δ_r and n_f such that `BIDIR-MATRIX-POWER` returns an estimate of desired accuracy. The choices of (n_f, δ_r) also affect the running time guarantee, which, as in Lemma **??**, is averaged over all targets $t \in [n]$. To this end, we define $T(t)$ to be the running time for a target index t. Now we have our main theorem:

Theorem 1: Given matrix Q and vector $\mathbf{z}$, let $\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t]$ be the estimate of $\mathbf{p}^\ell_{\mathbf{z}}[t]$ returned by the `BIDIR-MATRIX-POWER` algorithm with

$$\delta_r = \sqrt[3]{\frac{nnz(Q)\epsilon^2\delta^2}{\ell^2 n||\mathbf{z}||_1^2\beta^\ell\ln(\ell/p_{fail})}}$$

$$n_f = \frac{2\ell_{\max}^2|\mathbf{z}|_1^2\delta_r^2\beta^{2\ell_{\max}}}{\epsilon^2\delta^2}\ln\left(\frac{2}{p_{fail}}\right)$$

Then we have the following

- For all $t \in [n]$, we have $|\widehat{\mathbf{p}}^\ell_{\mathbf{z}}[t] - \mathbf{p}^\ell_{\mathbf{z}}[t]| \le \max\{\delta, \epsilon\mathbf{p}^\ell_{\mathbf{z}}[t]\}$ with probability at least $1 - p_{fail}$
- For a uniform random choice of $t \in [n]$, the expected running time of the algorithm is

$$\mathbb{E}[T(t)] = O\left(\left(\frac{||\mathbf{z}||_1 nnz(Q)}{\epsilon\delta n}\right)^{2/3}\ell^{5/3}\left(\ln\frac{2}{p_{fail}}\right)^{1/3}\right)$$

In particular, suppose we choose $\delta = 1/n$, and assuming $||\mathbf{z}||_1 \le 1$ and $||Q||_\infty \le 1$, and $\ell, 1/p_{fail} = O(1)$, then we get $\mathbb{E}[T(t)] = O\left((nnz(Q)/\epsilon)^{2/3}\right)$, as stated in Section **??**.

Proof: Consider the random walk-score $\mathcal{S}^\ell_t = ((\ell+1)/n_f)\sum_{i=1}^{n_f} S^\ell_{i,t}$ as in Algorithm **??**. From Lemma **??** we have that $\mathbb{E}\left[\mathcal{S}^\ell_t\right] = \mathbf{p}^\ell_{\mathbf{z}}[t] - \left\langle \sigma, \mathbf{q}^\ell_t\right\rangle$; moreover, from Lemma **??**, we have that $|\mathcal{S}^\ell_t| \le (\ell+1)||\mathbf{z}||_1\delta_r\beta^\ell$. Thus, to achieve relative error ϵ with probability at least $1 - p_{fail}$, we have from Lemma **??** that we need

$$n_f \ge \frac{2(\ell+1)^2|\mathbf{z}|_1^2\delta_r^2\beta^{2\ell_{\max}}}{\epsilon^2\delta^2}\ln\left(\frac{2}{p_{fail}}\right)$$

Given this choice, we know that the running time for the forward MCMC sampling is $O(n_f\ell)$; moreover, from Lemma **??**, we get that the reverse work running time for a uniform random choice of $t \in [n]$ is $\frac{nnz(Q)}{n\delta_r}(\ell+1)\beta^\ell$. Now substituting the value of δ_r given in the statement (which is chosen to balance the two running times), we get the promised running time guarantee. ■