A Dynamic Algorithm for Network Propagation

Barak Sternberg
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
barakolo@gmail.com
https://orcid.org/0000-0002-1803-6437

Roded Sharan[1]
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
roded@post.tau.ac.il

Abstract

Network propagation is a powerful transformation that amplifies signal-to-noise ratio in biological and other data. To date, most of its applications in the biological domain employed standard techniques for its computation that require $O(m)$ time for a network with n vertices and m edges. When applied in a dynamic setting where the network is constantly modified, the cost of these computations becomes prohibitive. Here we study, for the first time in the biological context, the complexity of dynamic algorithms for network propagation. We develop a vertex decremental algorithm that is motivated by various biological applications and can maintain propagation scores over general weights at an amortized cost of $O(m/n^{1/4})$ per update. In application to real networks, the dynamic algorithm achieves significant, 50- to 100-fold, speedups over conventional static methods for network propagation, demonstrating its great potential in practice.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Network propagation, Dynamic graph algorithm, protein-protein interaction network

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.7

Supplement Material `https://github.com/barakolo/dygraph_bio`

Funding R. S. was supported a grant from the Ministry of Science, Technology and Space of the State of Israel and the Helmholtz Centers, Germany.

1 Introduction

Network propagation has become a central technique in biology, as in other domains, to rank the relevance of genes to a process under investigation [7]. However, its complexity is becoming a bottleneck in dynamic settings where the network is subjected to multiple changes. In the biological domain, dynamic computations are essential not only because the network is updated with time but also because certain applications involve the systematic evaluation of propagation results under many network modifications. For example, in [12] the propagation over a network with n nodes is compared to n other propagations that are performed on modifications of the network where each time a different vertex is removed (simulating a knockout). Another application of the dynamic setting is when working with tissue-specific networks. As an example, in [14] multiple tissue-specific networks are formed from a given protein-protein interaction network by removing vertices with low expression, and propagation computations are applied to each.

[1] Corresponding author

While the computation of network propagation requires matrix inversion, a common and more efficient alternative utilizes the power iteration method [6]. This method approximates the propagation scores to within some additive constant at a cost of $O(m)$. An alternative local approach for obtaining approximate propagation was also suggested [3], yielding $O(m)$ time for a single propagation at worst case.

Focusing on a dynamic setting in which vertices are removed one by one [12], the total complexity of maintaining the propagation vectors after each removal becomes $O(mn)$ for n vertices when computing each propagation afresh. There is relatively scarce prior work regarding the computation of network propagation in a dynamic fashion which can be applied to the above setting. Specifically, Zhang et al. [5] and Ihsaka et al. [10] provide a fully dynamic propagation algorithm whose expected time per edge update is $O(1)$. However, both algorithms are limited to unweighted graphs and expected time analysis and might yield $O(mn)$ time under the settings considered here (of n vertex removals). This is true also for Yoon et al. [8], who provide a fully dynamic algorithm for network propagation but may lead to $O(mn)$ time under the settings considered here.

To tackle the dynamic computation challenge, we propose a novel algorithm that can handle general weights and several normalizations, including a symmetric normalization, a variant of which has been shown to be powerful in the biological domain [11, 12]. Our algorithm can handle n vertex removals in $O(mn^{3/4})$ total time. This yields a speedup of $\Omega(n^{1/4})$ over previous work. For real biological networks, this leads to a 50-fold to $100-fold$ speedup in the computations.

2 Preliminaries

We focus on undirected and weighted networks that represent protein-protein interactions. For a network G with n vertices and m edges, we denote by w the symmetric weighted adjacency matrix of the network. For a vertex u, we denote its set of neighbors by $N(u)$ and their number, i.e., the *degree* of u, by $d(u)$. The *weighted degree* $w(u)$ of u is the sum of weights of its adjacent edges. Two common normalizations of w to form a normalized matrix W are as follows: (i) normalizing each column of w to sum to 1, henceforth *weighted degree normalization*, and (ii) dividing each entry of w by the squared product of the weighted degrees of the corresponding nodes, i.e., $W_{ij} = w_{ij}/(\sqrt{w(i)}\sqrt{w(j)})$, henceforth *symmetric normalization* as used in [11].

Given a network G, a prior vector p of node relevance values (in [0,1]; typically 0 or 1), and a parameter $0 < \alpha < 1$, the *network propagation* transformation computes a score $s(v)$ for every node v that is a linear combination of its prior value and the average score of its network neighbors, reflecting its network proximity to the a-priori relevant nodes. Formally,

$$s(v) = \alpha p(v) + (1-\alpha) \sum_{u \in N(v)} s(u) W_{uv}$$

where $0 \leq \alpha \leq 1$ controls the tradeoff between prior information and network smoothing [7]. s, the propagation vector, can be computed analytically via matrix inversion or approximated using the "power iteration" algorithm which works as follows ([1, 9]):

1. Define $v_0 = p, i = 0$.
2. Compute $v_{i+1} = \alpha p + (1-\alpha) W v_i$.
3. While $i < \frac{\log(\epsilon)}{\log(1-\alpha)}$ increment i and goto (2).

The "power iteration" process is known to converge to the propagation vector ($\lim_{i\to\infty} v_i = s$), whenever the eigenvalues of W are at most 1 in absolute value, a condition satisfied

Algorithm 1 ForwardPush for weighted degree normalization.

```
1: procedure PUSH(v, R, P, α, W)
2:     P(v)+ = αR(v)
3:     for n ∈ N(v) do
4:         R(n)+ = (1 − α)R(v)W_vn
5:     R(v) = 0
6: procedure ForwardPush(W, α, ϵ, R, P)
7:     while ∃u|R(u)| > ϵd(u) do
8:         push(u, R, P, α, W)
9:     return P
```

by both normalizations presented above [7]. Here $\epsilon > 0$ is the required approximation bound on the sum of differences in absolute value (L1 error) between the computed and true propagation score. ϵ is typically chosen to be a constant smaller than 0.01. The following lemma characterizes the resulting propagation vector.

▶ **Lemma 2.1** ([2, 7]). *The propagation vector converges to* $\alpha(I - (1-\alpha)W)^{-1}p$.

Our algorithmic approach is motivated by the following lemma:

▶ **Lemma 2.2** ([6]). *Consider a random walk from prior distribution* p *using the weighted-degree normalized adjacency matrix* W*, where at each node* u *the walk stops with probability* α*. Then the total probability of the walk to stop at* u *is* $s(u)$*.*

3 The Forward-Push algorithm

In the following we describe our dynamic algorithms for the case of symmetric and weighted degree normalization. Our algorithm builds on the Forward-Push algorithm for the static case [3] which can be viewed as "simulating" random walks, “pushing” walks from one node to another and taking into account walks that stopped at any node, adding their probability to the node’s score. The algorithm is run until the residual walks have neglible effects (ϵ). For clarity, we fix the prior vector to p. The algorithm maintains two estimates per node u: the current estimate $P(u)$ of the probability to stop at u and the remaining probability $R(u)$ of walks that have reached u without stopping. The algorithm is given below and is called by initializing $P = 0^n$ (the zero vector) and $R = p$. For any nodes s, t we denote by $\pi(s,t)$ the score of t when propagating from s. Generalizing it, for any prior vector p and node u we denote by $\pi(p,u)$ the score of u when propagating from p. In the symmetric normalization case, $\pi(s,t) = \pi(t,s)$ for any two nodes s, t, while in the weighted degree normalization case it can be shown that $\pi(s,t)w(s) = \pi(t,s)w(t)$ (see, e.g., Lemma 1 in [13]). The following lemma is key in proving the correctness of the algorithm.

▶ **Lemma 3.1.** *The following equalities (algorithm’s invariants) are equivalent:*

1. $P(u) + \alpha R(u) = (1-\alpha)\sum_{x \in N(u)} P(x)W_{xu} + \alpha p_u$
2. $\pi(p,u) = P(u) + \sum_{x \in V} R(x)\pi(x,u)$

Proof. Denote $\pi = \alpha(I - (1-\alpha)W)^{-1}$. Note that $(\pi p)_u = \sum_{x \in V} \pi(x,u)p_x$, $(\pi R)_u = \sum_{x \in V} \pi(x,u)R(x)$. We can write (2) in vector form as: $\pi p = P + \pi R$. Multiplying both

sides by π^{-1} from the left we get: $p = \pi^{-1}P + R \leftrightarrow \alpha p = P - (1-\alpha)WP + \alpha R$. Rearranging terms we get the desired result. ◀

Using this lemma we can now justify the propagation approximation achieved by the ForwardPush algorithm.

▶ **Lemma 3.2.** *The ForwardPush algorithm maintains the invariants in 3.1.*

Proof. It suffices to prove that the first invariant holds. On initialization, $P = 0^n$ and $R = p$, hence the invariant holds. Suppose an arbitrary node u is pushed and denote by P', R' the updated values of P and R. Since $P'(u) = P(u) + \alpha R(u)$ and $R'(u) = 0$, we have $P'(u) + \alpha R'(u) = P(u) + \alpha R(u)$. Clearly, the right hand side of invariant (1) did not change for u as for any $x \in N(u)$ only $R(x)$ is changed while pushing u. For any other node x that is adjacent to u, $R'(x) = R(x) + (1-\alpha)R(u)W_{xu}$. Thus, the addition to the left hand side of the equation is $\alpha(1-\alpha)R(u)W_{xu}$ which is exactly the addition to the right hand side due to the update of $P(u)$. ◀

▶ **Lemma 3.3.** *For weighted-degree normalization, when ForwardPush() terminates:*

$$\sum_{t \in V} |P(t) - \pi(p,t)| \leq 2\epsilon m$$

Proof. Consider any node t and let e_t be the unit vector with 1 at coordinate t and 0 elsewhere. It follows from lemma 2.2 that for any node $u \in V$, $\sum_{t \in V} \pi(u,t) \leq 1$. By Lemma 3.1 and since $|R|_\infty \leq \epsilon$ upon termination,

$$\sum_{t \in V} |P(t) - \pi(p,t)| = \sum_{t \in V} \sum_{u \in V} |R(u)| \pi(u,t) \leq \sum_{t \in V} \sum_{u \in V} \epsilon d(u) \pi(u,t) = \sum_{u \in V} \epsilon d(u) \sum_{t \in V} \pi(u,t) \leq \sum_{u \in V} \epsilon d(u) = 2\epsilon m$$

◀

Note that the above lemma bounds the overall L1 approximation of the algorithm at $2\epsilon m$, hence ϵ is typically chosen to be smaller than $1/m$ to guarantee a constant overall error. After proving the correctness of the algorithm, we turn to bound its complexity:

▶ **Lemma 3.4.** *Every push of the algorithm, $|R|_1$, the sum of residuals in absolute value, decreases by at least $\alpha\epsilon d(v)$ where v is the node being pushed.*

Proof. Let R, R' denote the residuals before and after the push, respectively. Then

$$\sum_{u \in V} |R(u)| - \sum_{u \in V} |R'(u)| = |R(v)| + \sum_{u \in N(v)} |R(u)| - |R'(u)| \geq \quad (1)$$

$$|R(v)| - \sum_{u \in N(v)} |R'(u) - R(u)| = |R(v)| - (1-\alpha)|R(v)| \sum_{u \in N(v)} |W_{vu}| \geq \quad (2)$$

$$|R(v)| - (1-\alpha)|R(v)| = |R(v)|(1-(1-\alpha)) = \alpha|R(v)| \geq \alpha\epsilon d(v) \quad (3)$$

◀

▶ **Lemma 3.5.** *The ForwardPush algorithm with weighted degree normalization takes $O(\frac{|p|_1}{\alpha\epsilon})$ time.*

Proof. On initialization $\sum_{v \in V} |R(v)| = |p|_1$. Denote by u_i the vertex being pushed at step i of the algorithm. By Lemma 3.4, $\sum_{v \in V} |R(v)|$ decreases by at least $\alpha\epsilon d(u_i)$ at step i, thus the following holds: $\sum_i \alpha\epsilon d(u_i) \leq |p|_1$. Hence, $\sum_i d(u_i) \leq \frac{|p|_1}{\alpha\epsilon}$. As each push step of a node u_i can be implemented in $O(d(u_i))$ time, the total time is $O(\frac{|p|_1}{\alpha\epsilon})$. The residuals with absolute values greater than $\epsilon d(u_i)$ can be maintained in a linked list and an associated array at the same cost. ◀

We can generalize the above algorithm to any similar matrix $W = L^{-1}SL$ where L is an invertible diagonal matrix, S is a stochastic matrix, i.e: $S = wD^{-1}$, and D is the diagonal weighted degree matrix. For example, choosing $L = D^{1/2}$ yields the symmetric normalization. Note that such matrices satisfy the convergence condition as their eigenvalues are the same as those of S. The generalization is based on the following lemma:

▶ **Lemma 3.6.** *(Stochastic Lemma) Let $\psi(W, p, \alpha)$ be the result of network propagation with matrix $W = L^{-1}SL$ over prior p. Then $\psi(W, p, \alpha) = L^{-1}\psi(S, Lp, \alpha)$.*

Proof. Denote by v_n and v'_n the propagation vectors on W and S, respectively, after the nth step of the propagation process. We prove by induction that for every n, $v_n = L^{-1}v'_n$. For the base case $v_0 = p$ and $v'_0 = Lp$ so the claim trivially holds. Suppose the claim is true for n, then: $v_{n+1} = \alpha p + (1-\alpha)Wv_n = \alpha p + (1-\alpha)L^{-1}SL(L^{-1}v'_n) = L^{-1}(\alpha(Lp) + (1-\alpha)Sv'_n) = L^{-1}v'_{n+1}$. ◀

4 A dynamic decremental algorithm for vertex removals

Lemma 3.6 naturally suggests an adaptation for the *ForwardPush* algorithm for any normalization matrix W which is similar to a stochastic matrix. For concreteness and clarity, we will present a dynamic decremental algorithm for removing nodes under weighted symmetric normalization as used in [12]. The main idea is first to maintain a valid propagation result for weighted-degree normalization, and on deletion, using lemma 3.6, we fix the invariant. If this leads to high residuals, we will re-push them over the graph locally, to fix the total scores. In the following we set $L = D^{1/2}$ for symmetric normalization and $p' = LP$. For a removal of any node v, denote by S', D' the resulting matrices, by w' the updated network weights and by π' the updated propagation scores. Further denote by Leaf$N(v)$ the set of neighbors of v of degree 1. W.l.o.g. v has at least one neighbor which is of degree greater than 1, otherwise the connected component of v will vanish after its removal. The algorithm is given below (Algorithm 2). *ForwardPushSym* computes the initial propagation. *FixRemoveEdge* handles edge removals and is used as a subroutine by *VertexRemoveProp* which handles the node removals.

The following lemmas, proved in the Appendix, establish the correctness and accuracy of the algorithm. Denote $\phi := \max_v\{\sqrt{w(v)}, 1/\sqrt{w(v)}\}$. In all our applications reported below, for both yeast and human, $\phi < 36$.

▶ **Lemma 4.1.** *When $ForwardPushSym(W, \alpha, \epsilon, p)$ returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w(t)}} - \pi(p, t)| < 2\epsilon m\phi$ where $\pi(p, t)$ is the propagation score for node t with prior p under symmetric normalization.*

▶ **Lemma 4.2.** *When $VertexRemoveProp(v, R, P, W, \epsilon)$ returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w'(t)}} - \pi'(p, t)| < 2\epsilon m\phi$ where $\pi'(p, t)$ is the propagation score for node t with prior p under symmetric normalization after node v removal.*

Algorithm 2 ForwardPush for symmetric normalization.

1: **procedure** $ForwardPushSym(W = D^{-1/2}SD^{1/2}, \alpha, \epsilon, p)$
2: $R \leftarrow D^{1/2}p, P \leftarrow 0^n$
3: $R, P \leftarrow ForwardPush(S, R, P, \alpha, \epsilon)$
4: **return** R, P
5: **procedure** $FixRemoveEdge(u, v, R, P, W = D^{-1/2}SD^{1/2})$
6: $w(v), w_u(v) := \sum_{x \in V} w_{xv}, \sum_{x \in V, x \neq u} w_{xv}$
7: $F_u(v) = \frac{w(v)}{w_u(v)}$
8: $R(v) += \frac{1}{\alpha}(1 - \frac{1}{F_u(v)})P(v) - \frac{(1-\alpha)}{\alpha}P(u)S_{uv} + (\sqrt{w_u(v)} - \sqrt{w(u)})p_v$
9: $P(v) /= F_u(v)$
10: Remove (u, v) from S and normalize v.
11: **procedure** $VertexRemoveProp(v, R, P, W = D^{-1/2}SD^{1/2}, \epsilon)$
12: **for** $u \in \text{Leaf}N(v)$ **do**
13: $R(u) = 0, P(u) = \alpha p_u$
14: **for** $u \in N(v) \backslash \text{Leaf}N(v)$ **do**
15: $FixRemoveEdge(v, u, R, P, W)$
16: $R(v) = 0, P(v) = \alpha p_v$
17: $ForwardPush(S', R, P, \alpha, \epsilon)$
18: **return** R, P

Algorithm 3 Multiple-vertex removal

1: **procedure** $RemoveNodes(node\_list, W = D^{-1/2}SD^{1/2}, \alpha, \epsilon_0, \epsilon_1, p)$
2: $R, P \leftarrow ForwardPushSym(W, \alpha, \epsilon_0, p)$
3: **for** $u \in node\_list$ **do**
4: Backup R,P
5: $R, P \leftarrow VertexRemoveProp(u, R, P, W, \epsilon_1)$
6: $res(u) \leftarrow D^{-1/2}P$
7: Restore R,P
8: **return** res

Given the vertex removal routine, we can perform n removals and corresponding propagations using Algorithm 3. For efficiency, we can choose different accuracies ($\epsilon_0, \epsilon_1 = \epsilon$) for different stages of the propagation with $\epsilon_0 \leq \epsilon_1$. This allows us to invest more time in the initial propagation in order to reduce the resulting residual sum, leading to time savings in subsequent computations. In order to bound the complexity of the algorithm we denote by R^u the vector of residuals after fixing invariants in $VertexRemoveProp$ (line (16) completion) and define the total sum in absolute value of residual changes following the removal of a vertex u:

$$\Delta R_u := \sum_{v \in V} |R^u(v) - R(v)|$$

▶ **Lemma 4.3.** *The total residual sum being pushed over n vertex removals is bounded by:*

$$T := \sum_{u \in V} (\sum_{v \in V} |R(v)| + \Delta R_u)$$

Proof. The total sum of residuals being pushed when removing a vertex u is bounded by

$\sum_{v\in V\setminus\{u\}} |R^u(v)|$. By the triangle inequality, $\sum\limits_{v\in V\setminus\{u\}} |R^u(v)| \leq \sum\limits_{v\in V} |R(v)| + \Delta R_u$. ◀

▶ **Corollary 4.4.** *The removal of n vertices and subsequent propagations take $O(\frac{T}{\alpha\epsilon_1})$ time.*

▶ **Lemma 4.5.** *Let u be a node being removed, and $F_u(v)$ the expression described in Algorithm (2). Then,*

$$\Delta R_u \leq R(u) + \sum_{v\in LeafN(u)} R(v) + \sum_{v\in N(u)\setminus LeafN(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v +$$
$$(\frac{1-\alpha}{\alpha})P(u) \sum_{v\in N(u)\setminus LeafN(u)} S_{uv} + \frac{1}{\alpha} \sum_{v\in N(u)\setminus LeafN(u)} P(v)(1 - \frac{1}{F_u(v)})$$

Proof. Note that $R(v), P(v)$ are positive for any v as they represent the initial ForwardPush result. Assume that u is removed and consider the changes to its neighbors in VertexRemove(). Clearly, for each leaf neighbor v, the change in residuals is $R(v)$. For non-leaf neighbors v, we can upper bound the residual change by:

$$\frac{1}{\alpha}\left(1 - \frac{1}{F_u(v)}\right) P(v) + |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v + \frac{1-\alpha}{\alpha} P(u) S_{uv}$$

Hence, summing over all of these neighbors and u itself leads to the required result. ◀

To bound the complexity of the algorithm we need the following notation and auxiliary lemmas.

▶ **Lemma 4.6.** $1 - \frac{1}{F_u(v)} = S_{uv}$

Proof. $1 - \frac{1}{F_u(v)} = \frac{\sum\limits_{x\in V} w_{xv}}{\sum\limits_{x\in V} w_{xv}} - \frac{\sum\limits_{x\neq u} w_{xv}}{\sum\limits_{x\in V} w_{xv}} = S_{uv}$ ◀

▶ **Lemma 4.7.** *After the initial ForwardPushSym, $\sum\limits_{v\in V} P(v) \leq \phi$*

Proof. Initially, $\sum_{v\in V} P(v) = 0$ and $\sum_{v\in V} |R(v)| = |p'|_1$. Anytime we increase $P(u)$ by $\alpha R(u)$ for some u, $\sum_{v\in V} |R(v)|$ is reduced by at least $\alpha|R(u)|$ by Lemma 3.4. Therefore, $\sum_{v\in V} |R(v)| \leq |p'|_1$ throughout, implying that $\sum\limits_{v\in V} P(v) \leq |p'|_1$.

By definition, $|D^{1/2}|_1 = \sup\limits_{v\neq 0} \frac{|D^{1/2}v|_1}{|v|_1}$. Hence, $|p'|_1 = |D^{1/2}p|_1 \leq \frac{|D^{1/2}p|_1}{|p|_1}|p|_1 \leq |D^{1/2}|_1|p|_1 \leq \phi$. ◀

▶ **Lemma 4.8.** $\sum\limits_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| \leq \sqrt{w(v)}$

Proof. First, note that:

$$|\sqrt{w_u(v)} - \sqrt{w(v)}| = |\frac{(\sqrt{w_u(v)} - \sqrt{w(v)})(\sqrt{w_u(v)} + \sqrt{w(v)})}{\sqrt{w_u(v)} + \sqrt{w(v)}}|$$
$$= \frac{|w_u(v) - w(v)|}{|\sqrt{w_u(v)} + \sqrt{w(v)}|} \leq \frac{w_{vu}}{\sqrt{w(v)}}$$

Then, summing over all neighbors of v:

$$\sum_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| \leq \sum_{u\in N(v)} \frac{w_{vu}}{\sqrt{w(v)}} = \sqrt{w(v)}$$ ◀

▶ **Lemma 4.9.** $\sum_{u\in V} \Delta R_u \leq \frac{5\phi}{\alpha}$.

Proof. Let R, P be the updated residuals and the estimates after the initial application of ForwardPush() in line 2 of the RemoveNodes() algorithm. By Lemma 4.5, the total changes to the residuals are

$$\overbrace{\sum_{u\in V} R(u) + \sum_{u\in V}\sum_{v\in \text{Leaf}N(u)} R(v)}^{\text{part 1}} + \overbrace{\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v}^{\text{part 2}} +$$
$$\overbrace{\sum_{u\in V}(\frac{1-\alpha}{\alpha})P(u)\sum_{v\in N(u)\setminus \text{Leaf}N(u)} S_{uv}}^{\text{part 3}} + \frac{1}{\alpha}\overbrace{\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} P(v)(1-\frac{1}{F_u(v)})}^{\text{part 4}}$$

We will bound each part separately.

1. By definition, $\sum_{u\in V} R(u) \leq |D^{1/2}p|_1$. For a leaf $v \in V$, its residual will be summed once as its degree is 1. Overall, the sum is bounded by $2\sum_{u\in V} R(u) \leq 2|D^{1/2}p|_1 \leq 2\phi$.
2. By changing the summation order we get
$$\begin{aligned}\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v &= \\ \sum_{v\in V,\text{ not a leaf}} p_v \sum_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| &\overset{4.8}{\leq} \\ \sum_{v\in V,\text{ not a leaf}} p_v\sqrt{w(v)} &\leq |p|_1\phi = \phi\end{aligned}$$
3. we bound this part and get:
$$\sum_{u\in V}(\frac{1-\alpha}{\alpha})P(u)\sum_{v\in N(u)} S_{uv} \leq \frac{(1-\alpha)}{\alpha}\sum_{u\in V} P(u) \overset{4.7}{\leq} \frac{(1-\alpha)}{\alpha}\phi$$
4. Similar to part (1), by changing the order of summation we get that:
$$\begin{aligned}\frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v)\sum_{u\in N(v)}(1-\frac{1}{F_u(v)}) &\overset{4.6}{=} \\ \frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v)\sum_{u\in N(v)} S_{uv} &\leq \\ \frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v) &\leq \phi/\alpha\end{aligned}$$

Overall we obtain a bound of $\frac{5\phi}{\alpha}$. ◀

We are now ready to state our main result:

▶ **Lemma 4.10.** *The total complexity of the propagation algorithm with n vertex removals is* $(\frac{\phi}{\alpha\epsilon_0} + \frac{mn\epsilon_0}{\alpha\epsilon_1} + \frac{\phi}{\epsilon_1}) = O(m\sqrt{n\phi})$

Proof. The initialization takes $O(\phi/(\alpha\epsilon_0))$ time. By Lemmas 4.3 and 4.9, the total time for n vertices removals and subsequent propagations is bounded by:

$$\frac{\sum_{u\in V}(\sum_{v\in V}|R(v)|) + \Delta R_u}{\alpha\epsilon_1} \leq \frac{\sum_{u\in V}(\sum_{v\in V}\epsilon_0 d(v)) + \Delta R_u}{\alpha\epsilon_1} = \frac{\sum_{u\in V}(2m\epsilon_0) + \Delta R_u}{\alpha\epsilon_1} = \frac{2mn\epsilon_0}{\alpha\epsilon_1} + \frac{\frac{5\phi}{\alpha}}{\alpha\epsilon_1}$$

Table 1 Tested Networks and their properties.

Graph	#Nodes	#Edges	Maximal Degree	Average Degree	ϕ
Human (HIPPIE) [4]	19,796	339,788	2173	17.1	35.67
Yeast (ANAT [15])	5138	76,467	2040	13.82	19.9
Human (ANAT [15])	15,517	259,161	2086	15.73	27.75

Table 2 Performance evaluation on a human network (ANAT) upon 1,000 vertex removals.

Algorithm	DynamicFP	ForwardPush	Power Iteration
Time	10.1s	9417.89s	886.54s
#node visits per update	302.24	55,175,737	2,618,256
L1-error	0.00121	0.00119	0.00101

Hence, The complexity is:

$$O(\frac{\phi}{\alpha\epsilon_0}+\frac{mn\epsilon_0}{\alpha\epsilon_1}+\frac{\phi}{\epsilon_1}) \overset{\epsilon_0=\sqrt{\frac{\phi\epsilon_1}{mn}}}{=} O(\sqrt{\frac{mn\phi}{\epsilon_1}}) \overset{\epsilon_1=\Theta(\frac{1}{m})}{=} O(m\sqrt{n\phi})$$

The first equality follows by finding the minimal solution for ϵ_0, ϵ_1, where ϵ_1 is chosen to be c/m to bound the overall L1 error by some additive constant. As the edge weights are at most 1 and at least some positive constant (otherwise, the edges are considered unreliable), $\phi \leq \sqrt{n}$. Hence, the total time is $O(mn^{3/4})$ and the amortized cost per node knockout is $O(\frac{m}{n^{1/4}})$. ◀

5 Results

We benchmark our algorithm, Dynamic ForwardPushSym (DynamicFP), against a static implementation of the power iteration method, as well as against the static ForwardPush algorithm (ForwardPushSym from scratch). We measure the performance of each algorithm in terms of time (seconds) and the number of nodes it visits throughout its execution. In order to evaluate the accuracy of the different algorithms we measured their L1 deviation from the true propagation vector (sum of absolute differences). All tests were done in Intel(R) Xeon(R) E5410 @ 2.33Ghz, 16GB RAM. We used the latest yeast and human protein-protein interaction (PPI) networks from ANAT [15] as well as the human PPI network of [4]. The tested networks and their properties are summarized in Table 1. To perform a comparative analysis of different propagation algorithms, we fixed the propagation parameters to $\alpha = 0.4$ and a prior set size of 100. In order to compute accurate propagation scores against which we could benchmark the different methods, we applied the power iteration method with $\epsilon' = \epsilon/n$. In the comparison itself, we applied the power iteration method with the maximal ϵ that leads to an overall L1 error of at most 10^{-2}. For Forward-Push we used the maximal ϵ_1 that leads to an overall L1 error of at most 10^{-2}. We also set $\epsilon_0 = \sqrt{\frac{\epsilon_1\phi}{mn}}$. The results upon making 1,000 vertex removals, where each vertex is removed separately from the original network, are summarized in Tables 2 and 3. Next, we compared our performance to the previous approach LazyFwdUpdate of [5]. To this end, we used a larger human PPI network from [4] and observed the performance of the two methods upon knockouts of all (non-prior) vertices, simulating a real application of these methods. Since LazyFwdUpdate handles only unweighted networks with degree-based normalization,

Table 3 Performance evaluation on a yeast network (ANAT) upon 1,000 vertex removals.

Algorithm	DynamicFP	ForwardPush	Power Iteration
Time	2.44s	2132.2s	102.91s
#node visits per update	1443.904	22,103,498	765,310
L1-error	0.0013	0.00121	0.0017

Table 4 Performance comparison with $\epsilon_0 = \epsilon_1$ (1) and $\epsilon_0 = \sqrt{\frac{\epsilon_1 \phi}{mn}}$ (2).

Algorithm	LazyFwdUpdate	DynamicFP (1)	DynamicFP (2)
Time	36.01s	23.60	9.71s
#node visits per-update	13,925	8024	2749
L1-error	0.0047	0.0056	0.0061

we used the corresponding variant (unweighted network; weighted-degree normalization) of the *DynamicFP* algorithm. The results, summarized in Table 4, show that our algorithm compares favorably to LazyFwdUpdate, especially when varying the ratio between ϵ_0 and ϵ_1.

6 Conclusions

We have devised a dynamic algorithm for network propagation that can handle a vertex deletion in $O(\frac{m}{n^{1/4}})$ time and provides a speedup of $\Omega(n^{1/4})$ over previous work. Importantly, the algorithm leads to huge speedups of up to a 50-100 fold on real data. Thus, it allows, for the first time, the application of costly methods (such as [12] and [14]) to examine multiple gene knockouts simultaneously. The proposed algorithm can substantially increase the number of different knockout effects that can be simulated in a reasonable time. While our work focused on vertex deletions that are very common in the biological domain, it would be interesting to extend our algorithm to a fully dynamic one. This may result in various applications that concern situations in which edges rather than nodes are perturbed.

References

1 Amy N. Langville and Carl D. Meyer. Deeper Inside PageRank. *Internet Mathematics*, 1(3), jan 2004. doi:10.1080/15427951.2004.10129091.

2 Dengyong Zhou, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Scholkopf. Learning with Local and Global Consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, pages 321–328. MIT Press, 2004. URL: http://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf.

3 Glen Jeh and Jennifer Widom. Scaling Personalized Web Search. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, WWW '03, pages 271–279, New York, NY, USA, 2003. ACM. doi:10.1145/775152.775191.

4 Gregorio Alanis-Lobato, Miguel A. Andrade-Navarro, and Martin H. Schaefer. HIPPIE v2.0: enhancing meaningfulness and reliability of protein–protein interaction networks. *Nucleic Acids Research*, 45(D1):D408–D414, jan 2017. doi:10.1093/nar/gkw985.

5 Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate Personalized PageRank on Dynamic Graphs. *arXiv:1603.07796 [cs]*, 2016. arXiv: 1603.07796. URL: http://arxiv.org/abs/1603.07796.

6 Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web., nov 1999. URL: http://ilpubs.stanford.edu:8090/422/.

7 Lenore Cowen, Trey Ideker, Benjamin J. Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews. Genetics*, 18(9):551–562, 2017. doi:10.1038/nrg.2017.38.

8 Minji Yoon, WooJeong Jin, and U Kang. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. *arXiv:1712.00595 [cs]*, 2017. arXiv: 1712.00595. URL: http://arxiv.org/abs/1712.00595.

9 Monica Bianchini, Marco Gori, and Franco Scarselli. PageRank and Web communities. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pages 365–371, oct 2003. doi:10.1109/WI.2003.1241217.

10 Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient PageRank Tracking in Evolving Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, KDD '15, pages 875–884, New York, NY, USA, 2015. ACM. doi:10.1145/2783258.2783297.

11 Oron Vanunu, Oded Magger, Eytan Ruppin, Tomer Shlomi, and Roded Sharan. Associating Genes and Protein Complexes with Disease via Network Propagation. *PLOS Computational Biology*, 6(1):e1000641, 2010. doi:10.1371/journal.pcbi.1000641.

12 Ortal Shnaps, Eyal Perry, Dana Silverbush, and Roded Sharan. Inference of Personalized Drug Targets via Network Propagation. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 21:156–67, 2016. URL: https://www.semanticscholar.org/paper/Inference-of-Personalized-Drug-Targets-via-Network-Shnaps-Perry/b57104bd662ffeb95bb150d00adb381caffce013.

13 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Bidirectional PageRank Estimation: From Average-Case to Worst-Case. In *Algorithms and Models for the Web Graph - 12th International Workshop, WAW 2015, Eindhoven, The Netherlands, December 10-11, 2015, Proceedings*, WAW 2015, pages 164–176, New York, NY, USA, 2015. Springer-Verlag New York, Inc. doi:10.1007/978-3-319-26784-5_13.

14 Sushant Patkar, Assaf Magen, Roded Sharan, and Sridhar Hannenhalli. A network diffusion approach to inferring sample-specific function reveals functional changes associated with breast cancer. *PLoS Computational Biology 13(11): e1005793, in press*, 13, nov 2017. doi:10.1371/journal.pcbi.1005793.

15 Yomtov Almozlino, Nir Atias, Dana Silverbush, and Roded Sharan. ANAT 2.0: reconstructing functional protein subnetworks. *BMC bioinformatics*, 18(1):495, 2017. doi:10.1186/s12859-017-1932-1.

A Supplementary proofs

▶ **Lemma A.1.** *When ForwardPushSym(W, α, ϵ, p) returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w(t)}} - \pi(p,t)| < 2\epsilon m\phi$ where $\pi(p,t)$ is the propagation score for node t with prior p under symmetric normalization.*

Proof. By Lemma 3.3, *ForwardPush* applied on S with prior p' returns a vector P of

estimated scores, where $\sum_{t\in V}|P(t)-\pi(p',t)| = |P-\pi p'|_1 \le 2\epsilon m$.

$$\begin{aligned} |P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m \rightarrow \\ |P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m|D^{-1/2}|_1|D^{1/2}|_1 \rightarrow \\ |D^{-1/2}|_1|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m|D^{-1/2}|_1 \rightarrow \\ |D^{-1/2}|_1|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m\phi \end{aligned}$$

Since $|D^{-1/2}|_1 = \sup_{v\neq 0}\frac{|D^{-1/2}v|_1}{|v|_1}$ it follows that $|D^{-1/2}v|_1 = \frac{|D^{-1/2}v|_1}{|v|_1}|v|_1 \le |D^{-1/2}|_1|v|_1$. Therefore: $|D^{-1/2}P - D^{-1/2}\psi(S,D^{1/2}p,\alpha)|_1 \le 2\epsilon m\phi$.
Note that $\psi(W,p,\alpha)$ is a vector of propagation scores $\pi(p,t)$ for all nodes t. By the stochastic Lemma 3.6, $\psi(W,p,\alpha) = D^{-1/2}\psi(S,D^{1/2}p,\alpha)$. Thus,

$$|D^{-1/2}P-\psi(W,p,\alpha)|_1 \le 2\epsilon m\phi \rightarrow \sum_{t\in V}|\frac{P(t)}{\sqrt{w(t)}}-\pi(p,t)| \le 2\epsilon m\phi \quad \blacktriangleleft$$

▶ **Lemma A.2.** *When $VertexRemoveProp(v,R,P,W,\epsilon)$ returns, $\sum_{t\in V}|\frac{P(t)}{\sqrt{w'(t)}}-\pi'(p,t)| < 2\epsilon m\phi$ where $\pi'(p,t)$ is the propagation score for node t with prior p under symmetric normalization after node v removal.*

Proof. Denote by R',P',S',D' the changed residuals and estimates, weights and sum of degrees (for an isolated vertex u we define $D^{-1}_{uu} = D_{uu} = 1$), respectively, after line (16) in $VertexRemoveProp$ and $p' := D^{1/2}p$, $p'' := D'^{1/2}p$. Note that $p'_u := \sqrt{w(u)}p_u$, $p''_u := \sqrt{w_v(u)}p_u$ by definition of D' (after node removal). We will first prove the following:

▶ **Lemma A.3.** *The following holds:*
1. *if $x \neq v$ then $P'(x)S'_{xu} = P(x)S_{xu}$*
2. *if $x = v$ and $u \in N(v)$ then $P'(x)S'_{xu} = 0$*
3. *if $x = v$ and $u \notin N(v)$ then $P'(x)S'_{xu} = P(x)S_{xu}$*

Proof.
1. Let $x \neq v$, if $P(x) \neq P'(x)$ then $x \in N(v)$ by algorithm def, hence $P'(x) = P(x)/F_u(v)$ and $S'_{xu} = S_{xu}F_u(v)$. Then for sure $P'(x)S'_{xu} = P(x)S_{xu}$. If $P(x) = P'(x)$ then x is not a neighbor of v, hence by algorithm def. $S'_{xu} = S_{xu}$.
2. Let $x = v$ and $u \in N(v)$, by algorithm def, $S'_{xu} = S'_{vu} = 0$ as the edge was removed, and $P'(x)S'_{xu} = 0$.
3. Let $x = v$ and $u \notin N(v)$, by algorithm def, $P'(x) = P(x)$, as no edge that connects to u was removed, also $S'_{xu} = S_{xu}$ hence $P(x)S_{xu} = P'(x)S'_{xu}$. ◀

We will show that after line (16) in $VertexRemoveProp$ (2), the invariants of $ForwardPush$ (3.1) are kept with respect to p'' as the prior, S' as the weights matrix and α.

- Let $u \in \text{Leaf}N(v)$, then after line (16) node u becomes isolated and $P'(u) + \alpha R'(u) = \alpha p_v = \alpha p''_v$ as required.
- Let $u \in N(v)$, we will show the invariant is kept for u. We know:
$$P(u) + \alpha R(u) = (1-\alpha)\sum_{x\in V}P(x)S_{xu} + \alpha p'_u \tag{4}$$
We want to show:
$$P'(u) + \alpha R'(u) = (1-\alpha)\sum_{x\in V}P'(x)S'_{xu} + \alpha p''_u \tag{5}$$

Hence, it is enough to show that RHS of (5) minus the RHS of (4) equals to the LHS of (5) minus the LHS of (4).
Using Lemma A.3, for any node $x \neq v$ it holds $P'(x)S'_{xu} = P(x)S_{xu}$ and for $x = v$ it holds $P'(x)S'_{xu} = 0$. Then:

$$\begin{aligned}((1-\alpha)\sum_{x\in V} P'(x)S'_{xu} + \alpha p''_u) \quad &- \quad ((1-\alpha)\sum_{x\in V} P(x)S_{xu} + \alpha p'_u) = \\ -(1-\alpha)P(v)S_{vu} \quad &- \quad \alpha(p'_u - p''_u)\end{aligned}$$

By Algorithm 2, line 8:

$$\begin{aligned}(P'(u) + \alpha R'(u)) - (P(u) + \alpha R(u)) \quad &= \quad (P'(u) - P(u)) + \alpha(R'(u) - R(u)) \\ &= \quad -(1 - \frac{1}{F_v(u)})P(u) + \alpha[\frac{1}{\alpha}(1 - \frac{1}{F_v(u)})P(u) \\ &- \quad \frac{(1-\alpha)}{\alpha}P(v)S_{vu} + (\sqrt{w_v(u)} - \sqrt{w(u)})p_u] \\ &= \quad -(1-\alpha)P(v)S_{vu} + \alpha(p''_u - p'_u)\end{aligned}$$

- Let $u \notin N(v)$, using Lemma A.3, $P'(x)S'_{xu} = P(x)S_{xu}$ for any x. By algorithm def. $P'(u) = P(u)$, $R'(u) = R(u)$ and $p'_u = p''_u$. Then:

$$\begin{aligned}P'(u) + \alpha R'(u) \quad &= \quad P(u) + \alpha R(u) = \\ (1-\alpha)\sum_{x\in V} P(x)S_{xu} + \alpha p'_u \quad &= \quad (1-\alpha)\sum_{x\in V} P'(x)S'_{xu} + \alpha p''_u\end{aligned}$$

Therefore, after applying $VertexRemoveProp$ we compute a valid propagation vector over the weights S'' with p'' as prior and α after node v removal. using A.1 we get the same approximation. ◀