Local Update Algorithms for Random Graphs

Philippe Duchon, Romaric Duvignau

Local Update Algorithms for Random Graphs

Philippe Duchon [*1] and Romaric Duvignau [†1]

[1]Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France,
CNRS, LaBRI, UMR 5800, F-33400 Talence, France

June 30, 2014

Abstract

We study the problem of maintaining a given distribution of random graphs under an arbitrary sequence of vertex insertions and deletions. Since our goal is to model the evolution of dynamic logical networks, we work in a local model where we do not have direct access to the list of all vertices. Instead, we assume access to a global primitive that returns a random vertex, chosen uniformly from the whole vertex set. In this preliminary work, we focus on a simple model of uniform directed random graphs where all vertices have a fixed outdegree. We describe and analyze several algorithms for the maintenance task; the most elaborate of our algorithms are asymptotically optimal.

Keywords— random graphs, dynamic graphs, logical network maintenance, randomness preservation

1 Introduction

In decentralized networks and in particular in peer-to-peer networks[1] (P2P), the structure of the network is maintained locally by the nodes following a predefined network protocol. In most modern implementations (e.g., Gnutella [6], GNUnet [1], Freenet [3]), the structure of the network is highly dependent on the sequence of updates (node insertions or deletions), and as a consequence a malicious sequence of updates may result in a disconnected or badly structured network. Moreover, designing and analyzing the update algorithms defined by such network protocols is made harder by this dependence on the update sequence. In particular, analysis of the network evolution is typically performed under nicely behaving update schemes: new arrivals follow a Poisson process

[*]Philippe.Duchon@labri.fr
[†]Romaric.Duvignau@labri.fr
[1]Dynamic decentralized networks for sharing data or computing resources.

and living times follow an exponential distribution (used in the analysis of [8] and [10]), or some similar dynamicity hypothesis (as in [4] for bounding the mixing time of a Markov chain modeling the P2P network of [2]).

In order to avoid this dependence and to make the analysis of the network evolution somewhat easier, we propose local update algorithms that maintain *exactly* some probability distribution of random graphs, so that the distribution of the graph resulting from an arbitrary update sequence does not depend on the sequence itself, but only on some small parameter (ideally, only the number of nodes in the final graph).

This goal is very similar to what is achieved by the update algorithms for some randomized data structures such as randomized binary search trees [12, 9] or randomized skip lists [11]. An immediate benefit of using such an exact maintenance protocol lies in the analysis: only one probability distribution per network size has to be studied, and the analysis does not depend on some probabilistic model for the update sequence.

In this paper, we give a precise, general definition of what such a distribution preserving protocol should be, and illustrate the notion for a specific model of random graphs: uniform k-out graphs (simple digraphs with out-degree k); these graphs have good network properties associated with the uniform distribution. We describe and analyze several insertion and deletion algorithms that only work locally in the graph - global knowledge of the whole network is not assumed.

The problem of locally updating k-out graphs can be reformulated as follows: given a random uniform k-out graph G with n vertices, find a randomized procedure in order to insert (resp. delete) a vertex such that the resulting graph G' is a random uniform k-out graph with $n+1$ (resp. $n-1$) vertices. While we do not assume global knowledge of the whole network, our algorithms are assumed to know the *size* of the graph.

Our algorithms make extensive use of random sampling primitives, and need the ability to pick a uniform random vertex. Since we do not want to assume centralized knowledge of the whole graph, we restrict this ability to the use of a special primitive `RandomVertex()`, `RV` for short; we also use other random generation functions whose output distributions are known in advance and do not depend on the graph.

This `RV` function is somewhat implicitly assumed in many decentralized protocols in the literature when considering that a new node has to know some *friend* node in order to insert properly in the network, and is often referred to as an *external mechanism*. It is usually not explicitly required that such a friend node be uniformly distributed over the network, which is a strong assumption, but this hypothesis avoids any centralization of the network on a particular subset of the nodes. Similar mechanisms are used in the literature, e.g., hashing a name allows a node to contact a random node in Chord [13]; the protocol in [10] assumes a centralized server that caches a D-uniform subset of the nodes (D being a predefined constant); the random walks of [2] allow a new node to pick some uniformly distributed nodes during its insertion to build an almost random regular graph; the tokens used in the protocol of [5] also play the role of sampling uniform nodes in order to preserve the connectivity of the network

over time.

Such a procedure is often very costly as it consumes network bandwidth or momentarily breaks the decentralization (in particular in the server case). Hence we shall essentially measure the cost of an update algorithm by the number of times it needs to call the `RV` primitive during its execution. Our most effective algorithms are optimal in this regard: algorithm Ins2 asymptotically uses k calls to `RV`, and algorithm Del3 asymptotically uses $o(1)$ such calls (in expectation, for both algorithms).

The expected complexity of the update algorithms, when disregarding the cost of `RV`, should also be small; ideally it should not depend too much on the network size. This last criterion can be thought of as a safeguard in order to prevent the algorithm from saving calls to `RV` by exploring the entire graph (which, in the model we consider, is connected with asymptotic probability 1). The algorithms presented here have constant expected time.

The rest of this paper is organized as follows. In the next section, we define some notation, and give definitions for what we call distribution-preserving insertion and deletion algorithms; we also define precisely the random graph model we work with. Section 3 contains our major contributions; we describe and analyze our various distribution-preserving algorithms for insertion and deletion in random k-out graphs. This is followed by a conclusion, where we discuss some directions for further research. Due to space constraints, we omit all proofs that our algorithms are distribution-preserving.

2 Notation and models

2.1 Notation

All graphs we consider in this paper are simple, loopless directed graphs. For any such graph G and any vertex u, we note $N_G^+(u)$ for the outgoing neighborhood of vertex u, i.e., the set of vertices v such that (u, v) is an edge of the graph. The *closed* neighborhood, obtained by adding the vertex itself, is noted $N_G^+[u]$. Similarly, we note $N_G^-(u)$ for the incoming neighborhood of u: the set of vertices v such that (v, u) is an edge. We write $E|_{V'}$ for the edge set E restricted to the subset V' of the vertex set.

Throughout the paper, n stands for the size of the vertex set V.

When X is some random variable and ρ is some probability distribution, we write $X \sim \rho$ to mean that X follows the distribution ρ. To ease the notation and the reading, we take the liberty of noting the addition and removal of a singleton in an additive fashion, so that $S + x$ stands for $S \cup \{x\}$ and $S - x$ for $S \setminus \{x\}$.

2.2 Distribution preserving algorithms

We shall assume that all possible nodes of our graphs belong to some countable set Ω that we call our underlying *universe*. Such a set might be thought of as the

space of IP addresses for computers over the Internet, $\mathbb{N}^2$ for an approximation of some 2D geometric space, and so on. For any finite subset V of Ω, we note $\mathcal{G}_V$ the set of all simple digraphs with vertex set V and $\mathcal{G} = (\mathcal{G}_V)_{V \subset \Omega}$.

We now define what we consider to be valid update algorithms with regards to some family of probability distributions.

Definition Let $\mu = (\mu_V)_{V \subset \Omega}$ be a family of probability distributions such that each μ_V has support $\mathrm{Supp}(\mu_V) \subseteq \mathcal{G}_V$. A randomized algorithm $\mathcal{A}$ is a *μ-preserving insertion algorithm* (resp. deletion algorithm) if, for any V and any $u \in \Omega \setminus V$ (resp. $u \in V$), if $G \sim \mu_V$ then we have $\mathcal{A}(G, u) \sim \mu_{V+u}$ (resp. $\mathcal{A}(G, u) \sim \mu_{V-u}$).

Remark An equivalent formulation of the definition would be: $\mathcal{A}$ is a valid insertion algorithm (resp. deletion algorithm) if for any finite subset V of Ω, for any vertex u in $\Omega \setminus V$ (resp. in V), and for any graph g' in $\mathcal{G}_{V+u}$ (resp. $\mathcal{G}_{V-u}$), we have:

$$\sum_{g \in \mathcal{G}_V} \mu_V(g) \cdot \mathbb{P}(\mathcal{A}(g, u) = g') \;=\; \mu_{V'}(g')$$

with $V' = V + u$ (resp. $V - u$).

It is important to note, in the above definition, that although the input graph is assumed to be random, the vertex to be deleted[2] is not; the equations should hold for *any* deterministic choice of vertex. This is important because we want our results to be valid if this choice of vertices to be inserted or deleted is given to an adversary; still, because this choice of vertex is assumed to be deterministic, it cannot depend on the graph: one cannot, for instance, choose to delete one of the vertices with the highest indegree.

Explicit calculations show that a stronger model in which an adversary would be allowed to "see" the current graph, and then arbitrarily choose a vertex to be deleted, would only accept as a solution an algorithm that built a new graph independent of the previous one - essentially defeating our goal of maintaining dynamic graphs. The algorithms we present in this paper, working in a weaker adversary model, are significantly more efficient.

2.3 Random k-out graphs

We now introduce the simple model of directed graphs for which we will describe distribution preserving algorithms.

Definition A k-out graph is a simple directed graph with all vertices of out-degree exactly k.

We define $\mathcal{G}^k = (\mathcal{G}^k_V)_{V \subset \Omega, |V| \geq k+1}$, where $\mathcal{G}^k_V$ stands for the set of all k-out graphs having V as vertex set, as well as $\nu^k = (\nu^k_V)_{V \subset \Omega, |V| \geq k+1}$, where ν^k_V is the uniform distribution over $\mathcal{G}^k_V$. Clearly, $\mathcal{G}^k_V$ is empty if V has fewer than $k+1$

[2] Or inserted; but this does not matter with the distribution considered here.

elements; accordingly, the behavior of our deletion algorithms is undefined if they are applied to a graph of size exactly $k+1$.

Remark A random graph $G = (V, E) \sim \nu_V^k$ iff for all $v \in V$, $N_G^+(v)$ is uniformly distributed over the k-subsets of $V - v$ and the $N_G^+(v)$ are mutually independent. Notice that, for all $u, v \in V$, $N \subset V - v$ with $|N| = k$, $\mathbb{P}(N_G^+(v) = N) = 1/\binom{n-1}{k}$; also, $\mathbb{P}((u, v) \in E) = k/(n-1)$, and for any family $(u_i, v_i)_{1 \leq i \leq \ell}$ of such potential pairs, the corresponding events are mutually independent if and only if all u_i are distinct. Consequently, in a ν_V^k-distributed graph with n vertices, the indegree of any one vertex follows the binomial distribution with parameters $n-1$ and $k/(n-1)$.

2.4 From centralized to decentralized algorithms

We describe all our algorithms as centralized, sequential algorithms. It should be noted that they have some "local" features. The deletion algorithms only need to examine the preexisting graph G up to distance 2 (measured in the underlying undirected graph) from the vertex u to be removed (we need to examine the outgoing edges from predecessors of u). Also, the algorithms end up only removing edges that were incident to u, and adding new edges to predecessors of u. Similarly, our insertion algorithms only need to examine the direct neighborhoods of those vertices returned by calls to `RV`, or, in the case of algorithm Ins2, neighborhoods of vertices along short paths from vertices obtained through `RV`.

In light of this, it should be clear that our algorithms could easily be transposed into a decentralized, message-passing model where each vertex corresponds to a process that has access to its predecessors and successors. In this, we assume that knowing the "identity" of a vertex is sufficient to allow direct communication with the corresponding process – and that we are somehow given access to a `RV` primitive. The question of running such algorithms in an unreliable network, or concurrently, is way beyond the scope of our work.

3 Distribution preserving algorithms for k-out graphs

3.1 Basic random samplers

We give a precise description in this short section of the various random samplers used by our algorithms. We make a clear distinction between those that do not use `RV` (called *internal*) and those that do (*external*).

3.1.1 Internal procedures

Since we are concerned about minimizing the number of calls to `RV`, we allow our algorithms to use an alternate source of randomness. The following standard probability distributions can be used by our algorithms:

- `Bernoulli`(p) returns 1 with probability p and 0 with probability $1-p$.
- `Binomial`(n,p) returns $0 \leq j \leq n$ with probability $\binom{n}{j}p^j(1-p)^{n-j}$.
- `Uniform`(S) returns each element $x \in S$ with probability $1/|S|$.
- `Permute`(S) returns the elements of the set S in a random uniform order.

We also need once a subroutine `RandomVector`, not making use of `RV`, which, when given a random ordered ℓ-subset $(U_1, \ldots, U_\ell)$ of a set S and $m = |S|$, outputs ℓ independent uniform elements $V_1, \ldots, V_\ell$ of S, implemented as follows:

$X \leftarrow 0$
for $1 \leq i \leq \ell$ **do**
  if Bernoulli(X/m) **then**
    $V_i \leftarrow$ **Uniform**$(\{U_1, ..., U_X\})$
  else
    $X \leftarrow X + 1$
    $V_i \leftarrow U_X$

3.1.2 External procedures

Practically speaking, we would often like to sample an element from $V \setminus S$, where S is typically a *small* and *known* set. Whenever this is needed, we write `RVAvoiding(`S`)` – shortened `RVA` when S is deducible from the context – which is assumed to return a vertex from $V \setminus S$ instead of V. This is implemented using `RV` by calling the primitive until an element outside of S is sampled. The number of calls needed is geometrically distributed with expectation $n/(n-|S|)$, which is close to 1 when $|S| \ll n$.

In the algorithms, we will often only count $\mathcal{R}$, the number of calls to `RVA`, since if the set to avoid is bounded by some constant then the total expected number of times `RV` is called is always $\mathbb{E}(\mathcal{R})(1 + \mathcal{O}(1/n))$.

We extend the framework with another function `RandomSubset(r)` that returns a uniform random r-subset of V. It is implemented using `RVA`, starting with $W_0 = \emptyset$ and building iteratively a set W_i of size i avoiding W_{i-1}. Its asymptotic expected cost is r.

3.2 Insertion algorithms

3.2.1 Insertion with cost $2k$

We first introduce a natural insertion algorithm. When inserting a new vertex u, we sample a uniform k-subset of V for its outgoing neighborhood, and a random variable $L \sim$ `Binomial`$(n, k/n)$. Finally, a uniform L-subset of V is chosen as its incoming neighborhood. We then choose randomly, for each selected predecessor, one of its outgoing edges and *redirect* it towards u. While simple, this algorithm preserves the uniform distribution over k-out graphs.

Algorithm 1 INS1

Input: a digraph $G = (V, E)$, a vertex $u \notin V$
Output: a digraph G'

1: $V' \leftarrow V + u$; $n \leftarrow |V|$
2: $S \leftarrow$ **RandomSubset**(k)
3: $E' \leftarrow E \cup \{(u, v) \mid v \in S\}$
4: $L \leftarrow$ **Binomial**$(n, k/n)$
5: $W \leftarrow$ **RandomSubset**(L)
6: **for all** $v \in W$ **do**
7: $\quad X \leftarrow$ **Uniform**$(N_G^+(v))$
8: $\quad E' \leftarrow E' - (v, X) + (v, u)$
9: $G' \leftarrow (V', E')$

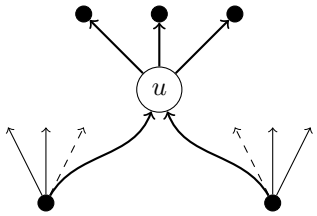

Figure 1: An execution of INS1 with $k = 3$ and $L = 2$.

Proposition 3.1 INS1 *is a ν^k-preserving insertion algorithm.*

Proposition 3.2 *The asymptotic expected cost of* INS1 *is $2k$.*

Proof We call only `RV` while processing `RandomSubset`(k) and `RandomSubset`(L), and each has an asymptotic expected cost of k (recall $\mathbb{E}(L) = k$). □

3.2.2 Insertion with cost k

We may improve the previous algorithm in order to get an expected cost of k, which is optimal in some sense (see Proposition 3.5). The idea is to reuse the neighbors of the vertices that will become the predecessors of u, the new inserted vertex. In the process, we have to take care not to introduce any dependencies between the outgoing neighborhoods.

Proposition 3.3 INS2 *is a ν^k-preserving insertion algorithm.*

Proposition 3.4 *The asymptotic expected cost of* INS2 *is k.*

Proof Let $\mathcal{R}$ be the number of times line 16 is processed during the execution of the algorithm. Except from this line, we call `RV` on line 3, while processing `RandomSubset` on line 22 and possibly other times if line 24 is executed. Notice this results in an expected cost of $k + \mathbb{E}(\mathcal{R})\,(1 + \mathcal{O}(1/n))$.

For $i \in \mathbb{N}$, let B_i be a Bernoulli random variable which is 1 if line 16 is executed during the ith run through the loop, and 0 otherwise (or if the loop has fewer than i runs); thus, $B_i = 1$ with probability bounded by $\frac{i}{n-(k-1)}\mathbb{P}(L \geq i)$. We have $\mathcal{R} = \sum_i B_i$; taking expectations,

$$\mathbb{E}(\mathcal{R}) \leq \sum_{i=1}^{n} \frac{i}{n-(k-1)}\mathbb{P}(L \geq i) = \frac{1}{n-(k-1)} \sum_{i=1}^{n} i\mathbb{P}(L \geq i).$$

Algorithm 2 INS2

Input: a digraph $G = (V, E)$, a vertex $u \notin V$
Output: a digraph G'

$V' \leftarrow V + u$; $E' \leftarrow E$; $n \leftarrow |V|$
$L \leftarrow \textbf{Binomial}(n, k/n)$
$X \leftarrow \textbf{RandomVertex}()$
$W \leftarrow \emptyset$
for $1 \leq i \leq L$ **do**
    $W \leftarrow W + X$
    $D \leftarrow \textbf{Uniform}(N_G^+(X))$
    $E' \leftarrow E' - (X, D) + (X, u)$
    if $i < L$ **then**
        $Y \leftarrow \{v \in N_G^+(X) - D \mid v \notin W\}$
        if $\textbf{Bernoulli}(|Y|/(n-i)) = 1$ **then**
            $X \leftarrow \textbf{Uniform}(Y)$
        else
            $X \leftarrow D$
            if $X \in W$ **then**
                $X \leftarrow \textbf{RVAvoiding}(W \cup N_G^+(X))$
    else
        if $\textbf{Bernoulli}(k/n) = 1$ **then**
            $X \leftarrow \textbf{Uniform}(N_G^+[X] - D)$
        else
            $X \leftarrow D$
$S \leftarrow \textbf{RandomSubset}(k-1)$
if $X \in S$ **then**
    $S \leftarrow S - X + \textbf{RVAvoiding}(S)$
$E' \leftarrow E' + (u, X) \cup \{(u, v) \mid v \in S\}$
$G' \leftarrow (V', E')$

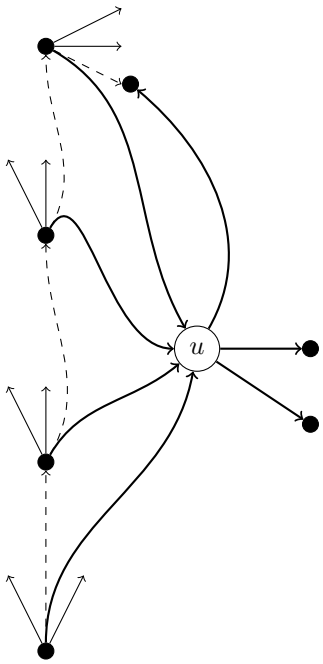

Figure 2: An execution of INS2 with $k = 3$ and $L = 4$.

Simple sum manipulations show that

$$\sum_{i=1}^{n} i\mathbb{P}(L \geq i) = \sum_{j=1}^{n} \frac{j(j+1)}{2}\mathbb{P}(L = j) = \mathbb{E}\left(\frac{L^2 + L}{2}\right).$$

Since L follows a binomial distribution with expectation k (hence variance less than k), this is bounded by $k^2/2 + k = \mathcal{O}(1)$. Thus $\mathbb{E}(\mathcal{R}) = \mathcal{O}(1/n)$. □

We conclude this section by arguing that algorithm INS2 is, in some sense, optimal.

The number of possible k-out graphs on a vertex set V of size n is $|\mathcal{G}_V^k| = \binom{n-1}{k}^n$. Taking ratios shows that there are $r_n = \binom{n}{k}(n/(n-k))^n$ times more such graphs with one more vertex. Thus, when given a uniform k-out graph on n vertices and asked to produce a uniform k-out graph on $n+1$ vertices, we need at least $\log_2(r_n) = k\log_2(n) + \mathcal{O}(1)$ additional bits of information. Each

Algorithm 3 DEL1

Input: a digraph $G = (V, E)$, a vertex $u \in V$
Output: a digraph G'

1: $V' \leftarrow V - u$
2: $E' \leftarrow E|_{V'}$
3: **for all** $v \in N_G^-(u)$ **do**
4: $\quad X \leftarrow$ **RVAvoiding**$(N_G^+[v])$
5: $\quad E' \leftarrow E' + (v, X)$
6: $G' \leftarrow (V', E')$

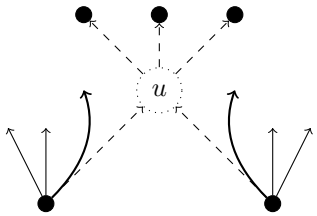

Figure 3: An execution of DEL1 with $k = 3$ and $L = 2$.

call to `RV` gives $\log_2(n)$ bits of information, since its result is uniform on a set of size n.

If `RV` were the only source of randomness used by our algorithms, then this would prove that no algorithm can use, in expectation, fewer than k such calls; but this is not the case. In fact, a single call to `RV` is enough to learn with high probability the whole vertex set by just exploring the connected component of a vertex, in the underlying undirected graph (which is connected with asymptotic probability 1). Subsequent calls can then be simulated once V is known. This would be at the cost of a much higher time complexity, though; our algorithm, in contrast, has constant expected time complexity if one assumes the calls to random samplers take constant time. Such an assumption is valid if one uses optimal samplers as described in [7].

Proposition 3.5 *For any ν^k-preserving algorithm $\mathcal{A}$ that has bounded expected time complexity and only makes calls in addition to `RV` to random samplers for distributions of bounded entropy, the expected number of calls to `RV` is, asymptotically, at least k.*

Proof The above discussion shows that fewer than k expected calls to `RV`, together with a ν^k-distributed graph of size n, and a bounded number of random numbers from distributions with bounded entropy, provide strictly less binary information than a ν^k-distributed graph of size $n + 1$. □

Note that our algorithm INS2 does respect the conditions of the above proposition: other than calls to `RV`, the only additional source of randomness is in the form of uniform variables on sets of size at most k (thus entropy at most $\log_2(k)$), Bernoulli variables (with entropy at most 1), and a single binomial variable with parameters n and k/n, whose entropy is close to that of the limiting Poisson distribution with expectation k. Thus, our algorithm is, at least in this class of algorithms, optimal.

3.3 Deletion algorithms

3.3.1 Simple deletion

We first introduce the following simple deletion algorithm : if u is leaving the network, this algorithm replaces any edge (v, u) pointing to u by an edge (v, x)

Algorithm 4 DEL2

Input: a digraph $G = (V, E)$, a vertex $u \in V$
Output: a digraph G'

1: $V' \leftarrow V - u$; $n \leftarrow |V|$
2: $E' \leftarrow E|_{V'}$
3: $u_1, \ldots, u_L \leftarrow N_G^-(u)$
4: $v_1, \ldots, v_k \leftarrow \mathbf{Permute}(N_G^+(u))$
5: $V_1, \ldots, V_k \leftarrow \mathbf{RandomVector}(v_1, \ldots, v_k, n-1)$
6: **for** $1 \leq i \leq L$ **do**
7: **if** $i \leq k$ and $V_i \notin N_G^+[u_i]$ **then**
8: $X \leftarrow V_i$
9: **else**
10: $X \leftarrow \mathbf{RVAvoiding}(N_G^+[u_i])$
11: $E' \leftarrow E' + (u_i, X)$
12: $G' \leftarrow (V', E')$

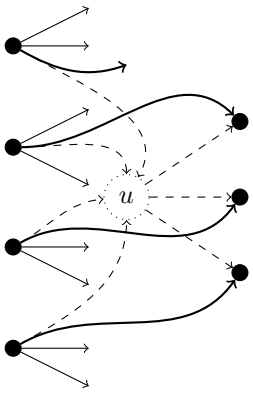

Figure 4: An execution of DEL2 with $k = 3$ and $L = 4$.

where x is chosen uniformly at random in the new vertex set, avoiding incompatible choices for x.

Proposition 3.6 DEL1 *is a ν^k-preserving deletion algorithm.*

Proposition 3.7 *The asymptotic expected cost of* DEL1 *is k.*

Proof In G, u has, in expectation, k predecessors. For each of them we make a call to `RVA`, resulting, in expectation, in $1 + \mathcal{O}(1/n)$ calls to `RV`. □

3.3.2 Improved deletion

We now describe a better deletion algorithm. The idea is to make use of the successors of the deleted vertex u as suggestions in order to save calls to `RV` for some of its predecessors; recall that $N_G^+(u)$ is a uniform random k-subset of $V - u$, independent of the other outgoing neighborhoods.

`RandomVector` is used to correct the dependencies between the successors of u. For the k first predecessors of u, we only call `RVA` if these suggestions cannot be accepted. In the algorithm description, we use the function `Permute` in order to guarantee that $N_G^+(u)$ is indeed in a random uniform order.

Proposition 3.8 DEL2 *is a ν^k-preserving deletion algorithm.*

Proposition 3.9 *The asymptotic expected cost of* DEL2 *is $e^{-k}\frac{k^k}{(k-1)!}$.*

Proof For $i \in \mathbb{N}$, let B_i be a Bernoulli random variable which is 1 if line 10 is executed during the ith run through the loop, 0 otherwise (in particular, if there is no such round). Notice $B_i = 1$ with probability $\frac{k-1}{n-1}\mathbb{P}(L \geq i)$ for $i \leq k$ and with probability $\mathbb{P}(L \geq i)$ if $i > k$.

Let $\mathcal{R} = \sum_{i\geq 1} B_i$ and notice the expected cost of DEL2 is simply $\mathbb{E}(\mathcal{R})(1 + \mathcal{O}(1/n))$, where

$$\begin{aligned}\mathbb{E}(\mathcal{R}) &= \sum_{i=1}^{n-1} \mathbb{E}(B_i) = \sum_{i=1}^{n-1} \left(\mathbb{1}_{i\leq k} \frac{k-1}{n-1} \mathbb{P}(L \geq i) + \mathbb{1}_{i>k} \mathbb{P}(L \geq i) \right) \\ &= \frac{k-1}{n-1} \sum_{i=1}^{k} \mathbb{P}(L \geq i) + \sum_{i=k+1}^{n-1} \mathbb{P}(L \geq i).\end{aligned}$$

The first term is $\mathcal{O}(1/n)$ since $\sum_{i=1}^{k} \mathbb{P}(L \geq i) \leq k$. Standard sum and binomial probabilities manipulations show that the second term may be rewritten as $k \cdot \mathbb{P}(L = k) + \mathcal{O}(1/n)$.

The result follows directly since $\mathbb{P}(L = k) = e^{-k} \frac{k^k}{k!} + \mathcal{O}(1/n)$. □

Remark The Stirling approximation formula implies that the asymptotic expected cost is bounded by, and close to, $\sqrt{\frac{k}{2\pi}}$.

3.3.3 Optimal deletion

In our previous algorithm, we improved on the natural deletion algorithm by recycling the successors of the deleted vertex u instead of "wasting" this information. Our last deletion algorithm, which has expected cost $o(1)$, uses u's predecessors instead of successors as suggestions to save calls to `RV`, and only one successor of u. Surprisingly, this can be done while preserving independence.

Proposition 3.10 DEL3 *is a ν^k-preserving deletion algorithm.*

Proposition 3.11 *The expected cost of* DEL3 *is $o(1)$.*

Proof We condition on the value ℓ of L, the number of predecessors of u in G. Notice that the probability that L is larger than, say, $n/2 - k$, is exponentially small, and, conditionally on such a large value of L, the expected number of calls to `RV` from all calls to `RVA` is still at most n^2 (at most n calls to `RVA`, each calling `RV` at most n times in expectation); thus, the contribution to the expected cost of such abnormally large values of L is exponentially small.

We now assume that $L < n/2 - k$. There are L possible calls to `RVA`. For $i < L$, the ith possible call does not occur if $u_{i+1} \notin N_G^+(u_i)$, i.e., the probability that it occurs is less than $(k-1)/(n-2)$. Since the forbidden set is always of size at most $\ell + k$, the ith potential call contributes at most $\frac{k-1}{n-2} \cdot \frac{n}{n-\ell-k} \leq 2\frac{k-1}{n-2}$ to the expected cost. Similarly, the Lth possible call occurs with probability $k/(n-1)$ and has a forbidden set of size $k+1$, resulting in an expected number of calls to `RV` equal to $\frac{k}{n-1} \cdot \frac{n}{n-k-1} = \mathcal{O}(1/n)$.

Thus, taking expectations for L – and keeping in mind that $\mathbb{E}(L | L < n/2) \leq \mathbb{E}(L) = k$, the total expected number of calls to `RV` is bounded above by

$$2\frac{(k-1)^2}{n-2} + \frac{nk}{(n-1)(n-k-1)},$$

which, for any fixed k, is $\mathcal{O}(1/n)$. □

Algorithm 5 DEL3

Input: a digraph $G = (V, E)$, a vertex $u \in V$
Output: a digraph G'

1: $V' \leftarrow V - u$; $n \leftarrow |V|$; $E' \leftarrow E|_{V'}$
2: $u_1, ..., u_L \leftarrow$ **Permute**$(N_G^-(u))$
3: **for** $1 \leq i \leq L$ **do**
4: **if** $i = L$ **then**
5: $X \leftarrow$ **Uniform**$(N_G^+(u))$
6: **if** $X \in N_G^+[u_L]$ **then**
7: $X \leftarrow$ **RVAvoiding**$(N_G^+[u_L])$
8: **else**
9: $C \leftarrow \{u_j \mid j < i \text{ and } u_j \notin N_G^+(u_i)\}$
10: **if Bernoulli**$(|C|/(n-1-k))$ **then**
11: $X \leftarrow$ **Uniform**(C)
12: **else if** $(u_i, u_{i+1}) \notin E'$ **then**
13: $X \leftarrow u_{i+1}$
14: **else**
15: $X \leftarrow$ **RVAvoiding**$(N_G^+[u_i] \cup C)$
16: $E' \leftarrow E' + (u_i, X)$
17: $G' \leftarrow (V', E')$

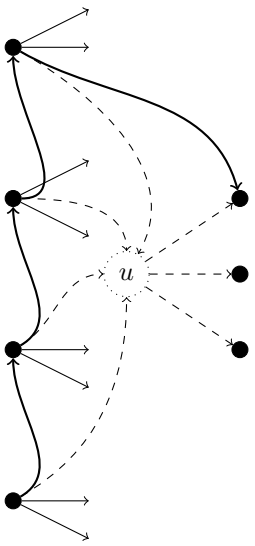

Figure 5: An execution of DEL3 with $k = 3$ and $L = 4$.

4 Conclusion and future research

We have defined a notion of distribution preservation for update algorithms on dynamic random graphs, and have illustrated it by giving examples of such distribution preserving insertion and deletion algorithms for uniform k-out graphs. Our algorithms are naturally suited to implementation in a distributed setting, once the problem of getting a random vertex in the graph is solved.

A natural direction for further research is to devise similar algorithms for other, more complex distributions on random graphs. Obtaining algorithms with similar properties for uniform undirected k-regular graphs (for even k, since for odd k such regular graphs do not exist for odd size) would be quite a feat, since known algorithms for sampling from this distribution tend to have poor time complexity for even moderate k. In the present work, the fact that the graphs are directed introduces a lot of independence that our algorithms rely upon.

Our k-out graphs are "simple" in that renaming the vertices in any (deterministic) way leaves the ν^k distribution unchanged. It would be interesting and challenging to apply the same ideas to random graph models where the "identities" of vertices actually have some influence over the distribution of graphs, such as any graph model that relies on vertices being placed in some geometric space.

References

[1] Krista Bennett, Tiberius Stef, Christian Grothoff, Tzvetan Horozov, and Ioana Patrascu. The gnet whitepaper. Technical report, Purdue University, 2002.

[2] Virgil Bourassa and Fred B. Holt. Swan: Small-world wide area networks. In *International Conference on Advances in Infracstructure, SSGRR-2003s*, 2003.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, pages 46–66, 2001.

[4] Colin Cooper, Martin Dyer, and Catherine Greenhill. Sampling regular graphs and a peer-to-peer network. *Comb. Probab. Comput.*, 16(4):557–593, 2007.

[5] Colin Cooper, Ralf Klasing, and Tomasz Radzik. A randomized algorithm for the joining protocol in dynamic distributed networks. *Theor. Comput. Sci.*, 406(3):248–262, 2008.

[6] Justin Frankel and Tom Pepper. The gnutella project. http://www.gnutelliums.com/.

[7] Donald E Knuth and Andrew C Yao. The complexity of nonuniform random number generation. In *Proceedings of Symposium on Algorithms and Complexity*, pages 357–428. Academic Press, New York, 1976.

[8] David Liben-nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *ACM Symposium on Principles of Distributed Computing*, pages 233–242, 2002.

[9] Conrado Martínez and Salvador Roura. Randomized binary search trees. *J. ACM*, 45(2):288–323, 1998.

[10] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter p2p networks. In *FOCS*, pages 492–499, 2001.

[11] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[12] Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996.

[13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, 2001.